PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

PROGRAMADORES

III, NÚMERO 35

975 Pts.

LINUX

Programación X-Window:
Driver SVGA para XFree 86

LIVECONNECT
Comunicación entre Java y JavaScript

CRIPTOGRAFÍA
De clave pública: RSA

REDES LOCALES
Seguridad en Novell Netware

JAVA
Interfaces, serialización y threads

DELPHI
Creación de componentes

Y ADEMÁS...

- RAD: Magic
- Las bases de datos en Visual Basic
- HP 9000, una nueva generación

CONTENIDO DEL CD-ROM

- LINUX X-WINDOW
- Plug-ing Liveconnect
- Visual JavaScript
- OCX y librerías Visual Basic
- TCL 7.6 y TK 4.2

8 413042 303299 00035

TOWER

SÓLO PROGRAMADORES

Número 35

es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es

Directora Adjunta
Amelia Noguera
anoguera@towercom.es

Colaboradores
Enrique Díaz, Arsenio Molinero, Carlos Álvaro, Enrique de la Lastra, Francisco José Abad Cerdá, Jose Antonio Collar, Javier Sarsa, Oscar Prieto Blanco, Ernesto Schmitz, Alejandro Reyero, Emilio Postigo Rian, Chema Álvarez, J. Antonio Mendoza.

Contenidos del CD-ROM
Chema Álvarez y Ernesto Schmitz

Jefe de Diseño
Fernando García Santamaría

Maquetación
Clara Francés

Tratamiento de Imagen
María Arce Giménez

Imagen de Portada
Fernando Núñez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
eriva@towercom.es
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Isabel Bojo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Filmación
Megatipo

Impresión
G. Reunidas

Distribución
SGEL

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló

Director Financiero
Francisco García Díaz de Liaño

Director de Producción
Carlos Peropadre

Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
COPYRIGHT AGOSTO 97

EDITORIAL

Inteligencia artificial

Cuando, a finales de los años sesenta, Stanley Kubrick y Arthur C. Clarke se unieron en su fabulosa odisea espacial y crearon una inteligencia artificial, HAL 9.000, muy humana en sus decisiones y patética en una muerte larga y dolorosa, estaban encendiendo el fuego de un debate que ha sido alimentado de forma creciente hasta nuestros días y ha alcanzado un auge formidable con la patética derrota (muy humana, larga y dolorosa) del campeón Kaspárov frente a la todavía fría Deep Blue. No es este el momento de seguir preguntándose si algún día Deep Blue pensará como un humano, esto es, si gemirá lastimosamente ante su inminente desconexión o asimilará nuestra idiosincrasia particular (esto es, ser capaz de freirse unos huevos mientras vemos la tele y se acuerda uno de su tía Packard o su abuelita IBM). No nos corresponde a nosotros, que sólo somos programadores, llegar a la explicación última de estos temas. Sí deben preocuparnos, en cambio, las ingentes y maravillosas posibilidades de este campo, sus amplias perspectivas específicamente programadoras.

Por este motivo, como os anunciábamos en el número anterior, desde aquí lanzamos un reto a todos aquellos que tengan el deseo irrefrenable de crear un ente inteligente, una tentación a la que han sucumbido desde Boris Karloff hasta Robert De Niro, pasando, por supuesto, por Arquímedes o Mary Shelley. En el interior de este número podréis ver las bases del concurso de programación de juegos de ajedrez, ese ancestral entretenimiento ante el que el ser humano ha sucumbido. El suculento premio que servirá de motivación a los participantes (una motivación, por lo demás, superflua, dado el intrínseco atractivo de la inteligencia artificial) es un Pentium Pro a 200 Mhz., con 64 Mb. de RAM, VIRGE4 Mb., lector de CD de 16 velocidades, disco duro de 5 gigas, monitor de 17 pulgadas, además de teclado, ratón, disquetera, etc. También habrá accesits para aquellos que se queden en las puertas de ganar. Los interesados podéis inscribiros en la dirección que incluimos en el interior de la revista. Desde aquí queremos animaros a todos a que participéis en el concurso. Ya sabemos, por los mensajes que hemos recibido en este mes, que la idea tiene gran aceptación entre vosotros, de modo que contamos con vuestra colaboración.

Por lo demás, este mes seguimos dando un papel destacado al lenguaje de programación Java. Se trata, como sabéis, del lenguaje del futuro, uno de los grandes pilares de Internet, y por ello es muy importante conocer sus características y sus posibilidades. También damos la enhorabuena a los que trabajen con Oracle, la versión 8 ya está aquí.

Tenéis a vuestra disposición la dirección de correo electrónico de la revista: **solop@towercom.es**, donde podéis enviar vuestras sugerencias sobre los contenidos, los temas que queráis que incluyamos y los productos que os gustaría que distribuyéramos con el CD-ROM, que todos los meses regalamos. Sois nuestros mejores colaboradores.

35 SÓLO PROGRAMADORES

25 Visual Basic Avanzado,
PROGRAMACIÓN OLE

Finalizando esta serie de cuatro artículos sobre pr gramación OLE con Visual Basic, veremos algun enfoques generales para utilizar esta interfaz de pr gramación en el diseño de aplicaciones. Diseñarem nuestro propio servidor.

29 Redes Locales
SEGURIDAD EN NOVELL

La seguridad es crucial en cualquier sistema opera tivo de red. Una administración correcta consigu evitar muchos quebraderos de cabeza. Conozca tod lo que necesita sobre la seguridad en Novell.

6 Noticias y libros
NOVEDADES DEL MERCADO

Todo sobre las nuevas herramientas de desarrollo Java, la versión actualizada del servidor HP con más futuro, oferta ampliada en sistemas abiertos de BULL, software de Sun para la red, etc.

11 WWW
LIVECONNECT: EL PUENTE ENTRE JAVA Y JAVASCRIPT

En esta sección dedicada al fascinante mundo de la red de redes, abordamos aspectos avanzados de la programación JavaScript a través del estudio de LiveConnect, como camino de comunicación hacia Java.

17 Criptografía
CRIPTOGRAFÍA DE CLAVE PUBLICA: RSA.

Este nuevo tipo de criptografía está basado en la teoría matemática de las funciones de un sólo sentido. Esta técnica apareció como un intento de mejorar la criptografía tradicional en dos de sus debilidades: la distribución de claves y el incremento de la probabilidad de ser descubierta. Estudiaremos todo sobre esta ciencia.

42 El PC por dentro
ANÁLISIS DEL PENTIUM PRO

El Pentium Pro está destinado, gracias a la reciente aparición del Pentium II, a convertirse en el substituto de todos los chips actuales, ya que posee muchas características mejoradas en todos los aspectos. Si quiere conocer mucho más sobre él, no se pierda este artículo.

38 Visual Basic
LAS BASES DE DATOS EN Visual Basic

Continuando con el curso sobre este entorno, veremos cómo operar con bases de datos casi sin utilizar la programación, gracias a los objetos de acceso a datos (Data Acces Object) que nos ofrece este potente lenguaje y su entorno, como los Recordsets, los Databases y los Querydef.

57 Programación en DELPHI
CREACIÓN DE COMPONENTES, TRUCOS Y NOCIONES IMPORTANTES

Veremos una serie de trucos para crear componentes de forma que cumplan la misión para la que fueron pensados, la reutilización de código. Un buen análisis previo de los requerimientos de cada uno de ellos es la mejor forma de comenzar.

65 Java
INTERFACES, SERIALIZACIÓN Y, DE POSTRE, TRHEADS.

En el artículo del mes anterior vimos las clases abstractas. En este artículo estudiamos en profundidad este tipo de clases y las interfaces, como técnicas de reutilización de código, y la serialización introducida en la última versión de Java.

75 Herramientas de desarrollo
RAD MAGIC 7

Las herramientas de desarrollo cada día tienen más importancia, se trata de programar lo mínimo posible, aunque la programación siga siendo inevitable. Analizaremos esta RAD que lleva tiempo en el mercado y que, basándose únicamente en tablas, sin una sóla línea de código, consigue que nos abstraigamos del entorno en el que se prevea que la aplicación tenga que funcionar.

48 Linux
PROGRAMACIÓN X-WINDOW: DRIVER SVGA PARA XFREE 86

Comenzamos una serie dedicada a la programación de drivers de vídeo para X-WINDOW. En el artículo que nos ocupa veremos la arquitectura de las SVGA y describiremos el proceso de construcción de un driver de víeo básico, para X-FREE 86.

78 Sistemas abiertos
SERVIDORES HP 9000

Recientemente Hewlett-Packard nos ha presentado un nuevo servidor que pretende revolucionar el tratamiento de los procesos de datos dentro de la empresa. Analizar este servidor es el objeto de este artículo.

81 CONTENIDO DEL CD-ROM

En esta ocasión pueden disfrutar del Gestor de ventanas LINUX X-WINDOW, plug-ings LiveConnect, OCX de VB, las últimas versiones de TCL y TK, tutoriales, Visual JavaScript y mucho más.

Noticias

NUEVAS EDICIONES DE SUPERCEDE
Aparecen las primeras herramientas completas RAD para el desarrollo de aplicaciones corporativas Java

SuperCede Java es un entorno de desarrollo para Java creado por Asymetrix Corporation para la creación de aplicaciones.

Gracias a este entorno, los programadores que utilizan Java como lenguaje de desarrollo pueden realizar todo tipo de aplicaciones de forma fácil, desde aquellas orientadas a la Web hasta las que utilizan habitualmente acceso a bases de datos.

En esta ocasión se presentan dos nuevas ediciones:

SuperCede Java/ActiveX Edition: con esta edición del entorno se pretende facilitar la construcción interactiva de forma visual de aplicaciones Java mediante la utilización de controles Java y ActiveX.

Este entorno incorpora la tecnología de conversión de TV Objects, con lo cual los diseñadores de Visual Basic pueden convertir los formularios que crean en este entorno en aplicaciones Java simplemente pulsando un botón.

Se permitetambién la modificación de los programas en tiempo de ejecución, lo cual resulta en un aumento de la productividad considerable.

SuperCede Database Edition: este entorno está más orientado a la creación de aplicaciones que utilizan bases de datos. Facilita el desarrollo mediante su fácil utilización. Añade al anterior entorno un soporte completo para el desarrollo y conectividad de las bases de datos. SuperCede Database Edition incluye dos implementaciones de JDBC (Java Database Connectivity): DataDirect de INTERSOLV y VisiChennel para JDBC de Visigenic Software.

Con esta edición se pretende proporcionar a los programadores una forma sencilla y rápida de crear aplicaciones de bases de datos Java sin utilizar la programación. También incluye una implementación de Java de una interfaz orientada al objeto para JDBC basado en la especificación ActiveX Data Objects (ADO) de Microsoft.

El producto se fundamenta en la versión del JDK 1.0.2 pero se prevé su actualización al 1.1 en breve, con la inclusión de mejoras orientadas hacia los nuevos elementos que incluye la actualización del kit de desarrollo que ofrece la empresa creadora de Java, Sun Microsystems.

La actualización a SuperCede 2.0 se podrá realizar de forma gratuita para los usuarios registrados de SuperCede.

BUL INCREMENTA SU OFERTA EN SISTEMAS ABIERTOS

Presenta la línea de productos UNIX: ESCALA y SAGISTER

Dentro de Escala se anuncian dos nuevas series: E y T con 2 procesadores y con 4, respectivamente, y hasta 4500 y 7300 tpm. Ambas series combinan el bajo coste del PC, con el bus interno PCI de 64 bits, con posibilidad de utilizar algoritmos RAID por hardware. Se han diseñado con arquitectura de 64 bits y su evolución a PowerPc 620 se puede realizar de forma transparente. En la gama Sagister incorporan nuevo software para la gestión y control de las aplicaciones cliente, la gestión de recursos del sistema, la regulación de la carga y la gestión de las prioridades de los procesos.

Bull defiende el nuevo microprocesaror PowerPC 604e a 200 Mhz que incorpora en Escala y Sagister, y anuncia un incremento de un 55% en su potencia. Como novedad se indica la disponibilidad de periféricos de bajo coste y alta capacidad y controladores de comunicaciones renovados.

La versión de SO incluida en los nuevos modelos es la 4.2 de AIX.

Para más información sobre el Grupo Bull consultar la dirección: http://www.bull.es.

SOFTWARE DE SUN MICROSYSTEMS PARA LA INFORMÁTICA DE WEB

Solaris, Solstice, Workshop y Security

Sun Microsystems está reforzando sus productos software orientados a integrar a la empresa en las soluciones informáticas basadas en la Web. De esta forma quiere ayudar a sus clientes a realizar la transición desde las redes tradicionales a las redes basadas en la Web. Los productos renovados han sido muy variados. Desde su entorno operativo Solaris, la aplicación Sun WorkShop, los entornos de desarrollo Java WorShop y el sistema de gestión de redes y seguridad Solstice.

Además la compañía ha incorporado la tecnología Java en todos los productos de software que ahora anuncia.

No nos sorprende la importancia que la mayoría de las empresas con peso en el sector le están brindando a la tecnología Java. Sun ha presentado recientemente el software Java Studio que incorpora un entorno visual de diseño personalizado que permite la creación de aplicaciones basadas en la Web. Como metodología de desarrollo se utiliza la integración de componentes reutilizables (Java Beans).

Para más información sobre la actualización del software, hablar con Ruth Utande de la empresa Intro Ibérica, en el teléfono 91 431 67 40.

Breves

SAS INSTITUTE LANZA SU NUEVO SOFTWARE PARA INTRANETS: INTRNET

SAS Institute es una compañía dedicada al desarrollo de herramientas software para análisis y extracción de datos. Recientemente ha anunciado la disponibilidad de un nuevo producto software para usuarios que dispongan de navegadores para Web.

SAS/*IntrNet* es un servidor de aplicación que presenta un mecanismo de distribución de la información muy accesible al usuario. Las aplicaciones del software de *IntrNet* son muy variadas, desde la publicación de páginas Web, ya que se puede utilizar para la creación de páginasWeb en un servidor Internet, su utilización como servicio dinámico de datos, sirviendo como sistema para responder a cuestiones planteadas por el usuario de forma interactiva, hasta su aprovechamiento como servicio dinámico de computación.

LAS PYMES ESPAÑOLAS PODRÁN CONSULTAR A TRAVÉS DE LA WEB SOBRE NUEVAS TECNOLOGÍAS

Microsoft pone a su disposición una página Web para facilitar a las pymes su integración en las nuevas tecnologías. En la elaboración de la página colaboran expertos del mundo empresarial con artículos sobre distintos temas financieros y tecnológicos.

La página quiere ser un foro de debate y análisis donde las pequeñas y medianas empresas puedan tener un punto más de información. Sus contenidos se actualizarán constantemente La web está disponible en la siguiente dirección:
http://www.microsoft.com/spain/pymes.

Nueva versión del software de producción Opencold+ DE EASTMAN SOFTWARE

El producto es una versión actualizada para Microsoft Windows NT 4.0. y complementa su software de producción para la gestión corporativa de procesos: OPENimage, OPENworkflow y OPENcold+.

Eastman Software Ibérica basa el funcionamiento de sus productos en los entornos cliente/servidor. Éstos pueden funcionar tanto en modo centralizado como en modo distribuido, a través de una red de dominios NT y como un servicio del servidor. La segunda parte de su nombre, COLD, son las siglas de *Computer Output to Laser Disk*, lo cual significa que el producto proporciona los medios para realizar el almacenamiento, la indexación y la recuperación de grandes volúmenes de datos, en formato de impresión (listados, facturas, albaranes, cheques, etc.).

OPENCold+ permite evitar la impresión de los listados, tiene capacidad para organizar toda la información que está contenida en los mismos, además consigue un almacenamiento electrónico de la información y facilita el proceso de búsqueda y recuperación de la información en los listados.

Para más información dispone del siguiente número de teléfono: (91) 308 43 15

OPENSERVER 5.0.4 Aparece la siguiente versión del sistema operativo de SCO

OpenServer 5.0.4 se une al boom de las tecnologías Internet, y para corroborarlo, incorpora Nestcape FastTrack Server 2.0, Nestcape Navigator Gold 3.0, el kit de desarrollo y la máquina virtual Java 1.1, SCO PP de Morning Star y SCO Internet configuration Manager.

La nueva versión del sistema operativo 5.0.4. se dirige a diversos mercados, está orientadotanto a pequeñas y medianas empresas como para servidores de aplicaciones, intranets y desarrolladores de aplicaciones, está diseñado para ser utiizado en Internet, e incorpora utilidades para facilitar la conectividad con Windows: SCO VisionFS 1.2 y SCO TermLite.

El software que integra el SCO OpenSErver 5.04. se presenta en cuatro configuraciones diferentes que se adaptan a las necesidades de sus clientes: Enterprise System, Internet FastStart, Host System y Desktop System. Estas configuraciones tienen en común los suplementos y actualizaciones integrados y un soporte adicional de hardware, hasta 4GB de memoria, discos duros IDE de hasta 8 GB), listas de nuevos drives, adaptadores de red y un nuevo comando de configuración de hardware mucho más fácil de utilizar. Como mejora añadida se han optimizado los puertos serie con una configuración más sencilla.

64 PROCESADORES CONECTADOS CON LOS SERVIDORES WINDOWS NT S1000 Microsoft y Tandem utilizan la tecnología de interconexión ServerNet

Ambas empresas han realizado esta demostración utilizando Windows NT server, enlazando 64 procesadores Intel Pentium Pro en un cluster con la tecnología de interconexión ServerNet de Tandem. El sistema gestionó una base de datos real de 2 terabytes con una tabla de 30.000 millones de filas. Y es que las ciencias avanzan que es una barbaridad y cuando no nos actualizamos a las últimas versiones por las buenas hay que hacerlo por las malas. Y qué mejor que hacer estas demostraciones de fuerza para convencernos.

Un cluster debe funcionar como un único ordenador si es SMP (multiproceso simétrico). Para su construcción se utilizó un sistema de 16 servidores Tandem S100 de cuatro procesadores Pentium Pro a 200 Mhz. Los servidores S1000 de Tandem están optimizados para explotar las ventajas de la tecnología ServerNet y como muestra un botón.

Para obtener más información sobre el tema consultar en el núméo de teléfono 3 50 04 14 (srta. Ana Velasco) o bien a través de la siguiente dirección de correo electrónico: firstlinepress@mad.servicom.es.

HEWLETT-PACKARD APUESTA POR HP 9000

La nueva gama de servidores empresariales se basa en micros de 64 bits

Estos nuevos servidores se han diseñado para hacer más fácil el crecimiento hacia mayores rendimientos en los procesos de transacciones (OLTP), los procesos de cálculo, el soporte a la toma de decisiones y entornos Internet. Con sus diferentes modelos de la clase K se intenta proporcionar una solución para el soporte de aplicaciones, bases de datos, redes de área local, sistemas de archivos de red y procesos de cálculo.

Dentro de la gama los servidores de Proceso Paralelo EPS 23 y EPS 40 implementan el proceso paralelo masivo y el multiproceso simétrico, mediante la arquitectura *Enterprise Parallel Server,* en el caso del servidor EPS 23. El servidor EPS 40 se basa en los ervidores empresariales V2200 de HP, y proporciona la escalabilidad que se necesita cuando se trabaja con tecnología de procesos paralelos masivos.

Esta gama soporta hasta 16 nodos de multiproceso simétrico paralelo en un sistema, y esta característica es importante porque permite que los usuarios puedan vincular cientos de procesadores para realizar la ejecución de aplicaciones. EPS 23 tiene un límite de 84 procesadores, mientras que el máximo de EPS 40 estaría en los 512 procesadores.

Para más información consultar el teléfono de Hewlett-Packard en Madrid: 91 6 31 16 00.

ÚLTIMOS SERVIDORES DE APPLE

Para grupos de trabajo e Internet/Intranet

Se basan en el procesador PowerPC 604e y en el sistema operativo Mac OS de Apple, y aportan nuevas funcionalidades a los grupos de trabajo en los mercados de edición profesional, educación y empresa. Se proporcionan tres soluciones software distintas a gusto del cliente que están diseñadas para cubrir distintas necesidades de servidor.

El Apple Workgroup server 9650/233 está orientado para clientes que necesitan utilizar intranets fáciles de usar y mantener. El producto incorpora un procesador PowerPC 604e a 233 Mhz (residente en una tarjeta de procesador intercambiable); 512 k de caché nivel 2, 64 MB de RAM (que son apmliables a 768 MB); tres canales SCSI; uno o dos discos duros Ultra/Wide de 4 GB (72009 rpm); un lector de CD-ROM de 12x; y, en una de las configuraciones, una unidad de cinta de audio digital (DAT) DDS-2 para backup.

El Apple Workgroup Sever 7350/180 tiene una alta capacidad de comunicación en red, y permite al usuario añadir capacidades complementarias instalando tarjetas de expansión. Con este producto se puede configurar con rapidez un completo servidor de aplicaciones, un servidor AppleShare o un servidor Internet/intranet, como variantes del software.

Para más información: http://www.apple.es o en el teléfono 902 11 2000.

Breves

La firma suiza de ingeniería EMCH+ Berger utiliza el estándar de software de Bentley

Los productos de Bentley serán el software utilizado para todos los proyectos de esta importante empresa suiza de ingeniería para el diseño de carreteras y sistemas de tráfico, ingeniería y construcción, Geoingeniería y planificación ambiental.

La elección del proveedor de software de ingeniería ha recaído en Bentley y la instalación y compra inicial de los productos y servicios Bentley asciende a un millón de dólares. En este presupuesto se incluyen licencias de MicroStation, MicroStation Modeller y MicroStation GeoGraphics.

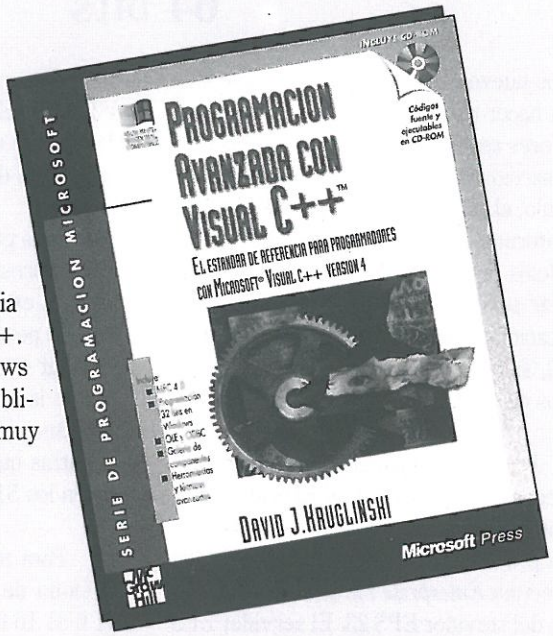
LA ONCE, la facultad de informática y empresas del sector realizan donaciones informáticas a centros especiales

El director general de la ONCE, Enrique Servando Sánchez, preside la Comisión Nacional de ayuda a centros de minusválidos, que se ha creado para conceder dotaciones informáticas a los centros de formación o instituciones que orientan sus actividades al mundo de los minusválidos. Bajo el auspicio de dicha comisión, en la que forman parte distintas personalidades del mundo informático, se ha realizado una primera entrega de recursos al Centro de Recursos Educativos de la ONCE Joan Amades, en Barcelona.

La donación consistió en un ordenador para multimedia y comunicaciones, de Packard Bell, y una impresora en color de Lexmark. Esperamos que las donaciones continúen para ayudar en la integración que la informática facilita a las personas discapacitadas.

LIBROS

PROGRAMACIóN AVANZADA CON VISUAL C++

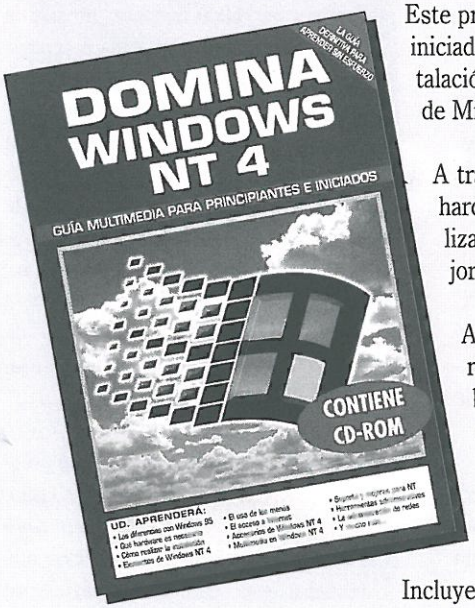

PROGRAMACION AVANZADA CON VISUAL C++™
EL ESTANDAR DE REFERENCIA PARA PROGRAMADORES CON MICROSOFT® VISUAL C++ VERSION 4
DAVID J. KRUGLINSKI
Microsoft Press
SERIE DE PROGRAMACION MICROSOFT

Dentro del tipo de libros a los que nos tiene acostumbrados McGraw Hill, encontramos éste sobre Visual C++. Destinado a programadores que conozcan el lenguaje C++, la obra conduce a través de temas como Windows, los aspectos fundamentales del entorno, la clase MFC Library View, la arquitectura documento-vista, OLE o el manejo de la base de datos, al conocimiento profundo del entorno y del sistema.

Programación avanzada en Visual C++es un material de referencia muy completo sobre todo lo relacionado con Microsoft Visual C++. Comprenderemos los dos sistemas de desarrollo de aplicaciones Windows que se integran en el entorno: el Win-32 API, y la programación con la blibioteca MFC para marcos de aplicaciones, de la cual se hace un estudio muy detallado en una obra de excelente calidad.

Editorial: McGraw Hill Microsoft Press
Autor: David J. Kruglinski.
Idioma: español
Incluye CD-ROM
Precio: 8.495 ptas.

DOMINA WINDOWS NT 4



Este producto es una guía multimedia para facilitar a los usuarios principiantes y a los ya iniciados el aprendizaje de Windows NT. Con *Domina Windows NT*, podrá dominar la instalación, configuración y el manejo de la última versión del sistema operativo de 32 bits de Microsoft.

A través de esta obra multimedia podrá aprender: las diferencias con Windows 95, el hardware necesario, todo sobre la instalación, los elementos de Windows NT 4, la utilización de los menús, el acceso a Internet, los accesorios de Windows NT 4 y las mejoras que esta versión consigue, administración de redes, etc.

A través del programa podrá llegar a los diferentes temas que le interesen de forma rápida, utilizando las herramientas de navegación como en la Web para facilitarle la búsqueda de la información. Adèmás dispone de un glosario integrado para explicarle en línea aquellos conceptos que desee. Una sección de consejos y trucos completan la obra.

Editorial: Abeto
Idioma: español
Incluye CD-ROM

LiveConnect: el puente entre Java y JavaScript

WWW

Alejandro M. Reyero Abad
axl@las.es

Todo el mundo envidia a los autores de las páginas WEB que, más que documentos, podrían pasar por obras de arte. El diseño gráfico que se necesita para la elaboración de las mismas requiere buen gusto, inspiración y dominio de programas de retoque gráfico. La programación que se utiliza en dicho mundo, en cambio, en principio no resulta muy complicada.

Nuestro objetivo a partir de este artículo y en los que le seguirán es conocer las técnicas y los trucos más utilizados y resultones.

En toda página WEB que se precie de estar a la última encontraremos algún applet Java o algunas gotas de JavaScript, lo que ya no resulta tan común es encontrar páginas en las que estos dos elementos, normalmente de mundos diferentes, convivan y se relacionen entre ellos en la misma página. Desde la versión 3.0 de Netscape esto se hace posible gracias a la tecnología *LiveConnect*.

Introducción a Liveconnect

LiveConnect hace posible la comunicación entre applets del lenguaje Java y JavaScript en una página y la comunicación entre JavaScript y los distintos plug-ins para Netscape.

De esta manera podremos usar JavaScript para acceder a variables, métodos, clases y librerías Java, controlar el comportamiento de applets y plug-ins y usar código en Java para acceder a métodos y funciones JavaScript.

Por defecto, *LiveConnect* está activado en Netscape 3.0 y superiores; para que funcione, tanto Java como JavaScript tienen que estar activados. Para comprobarlo nos cercioraremos de que las opciones *Enable Java y Enable JavaScript* se encuentren seleccionadas.

Si disponemos de Netscape 3.0, estas opciones se encuentran en la sección *Languajes*, dentro de *Network Preferences*, en el menú *Options*.

LiveConnect hace posible la comunicación entre código JavaScript y los applets de Java contenidos en una página WEB

En Netscape 4.0, están en *Advanced*, en edición de preferencias de *Edit*.

Después de conocer a fondo el modelo de objetos de JavaScript y aprender a utilizar el objeto window, estamos preparados para afrontar empresas más importantes. Este es el primer artículo de una serie, eminentemente práctica, sobre aspectos avanzados de la programación JavaScript. *LiveConnect* es el mejor comienzo posible.

También es conveniente abrir la consola Java (ya que vamos a utilizar applets). En esta consola aparecerán todos los mensajes que generan las variables de clase *err* y *out* del *package java.lang.System*. Para poder abrir la consola Java, hay que escoger *Show Java Console* en el menú *Options*, si tenemos Netscape 3.0, o *Java Console* en el menú *Communicator* en la versión 4.0 de Netscape.

La principal utilidad de la consola Java es el seguimiento del valor de las variables durante la ejecución de un applet para determinar posibles errores en nuestro código.

Un ejemplo de un applet que puede producir salida a través de la consola Java, podría ser el que reproducimos en las líneas de código que se muestran a continuación:

```
public void init() {
  System.out.println("Esto sale por la consola
                      Java")
```

Se puede utilizar también para presentar mensajes al usuario, pero muy pocos tienen la consola abierta por defecto.

Packages Netscape

Un *Package* no es más que una colección de clases Java. La versión 3.0 de Netscape Navigator contiene un fichero llamado *java_30* que incluye los *packages* de Netscape, que están dedicados a la comunicación entre el lenguaje Java y JavaScript, *java y sun.*

Los dos últimos no merecen más comentarios, puesto que se trata de los mismos que han sido incluidos dentro del JDK 1.0.2 de Sun (para conseguir más información, consúltense los artículos sobre Java de Oscar Prieto Blanco en *Sólo Programadores*).

El fichero *java_30*, que incluimos en el CD que acompaña la revista, engloba los *packages* Netscape que detallamos a continuación:

- *netscape.javascript*, que implementa la clase *JSObject* para permitir a los applets Java acceder a los métodos y propiedades JavaScript. También contiene la clase *JSException*, que provoca excepciones cuando encuentra un error en código JavaScript.

- *netscape.plugin*, que implementa la clase *Plugin* para permitir la comunicación entre JavaScript y los plug-ins Java. Si programamos un plug-in, deberemos compilarlo con esta clase para que los applets y nuestro código JavaScript puedan manipularlo.

- *netscape.applet*, es exactamente igual que el *sun.applet* del JDK de Sun, pero más seguro y, en principio, más rápido ya que se encuentra optimizado para Netscape.

- *netscape.net*, como el anterior, sustituyendo en esta ocasión al package *sun.net* del JDK de Sun.

Podemos utilizar estos *packages* también con el compilador del JDK, con la única variante de tener que añadir el fichero *java_30* a la variable de entorno *CLASSPATH*.

Comunicación de JavaScript a Java

Con *LiveConnect* tenemos la posibilidad de comunicarnos con Java de tres maneras diferentes:

- Para llamar métodos Java directamente.

- Para controlar applets Java.

- Para controlar plug-ins que se hayan programado en lenguaje Java.

• Accediendo a Java directamente

Una vez activado *LiveConnect*, podremos hacer llamadas directas a los métodos Java. Como vimos antes, podemos usar la consola Java para sacar mensajes desde un applet, pues bien, también podremos llamar a *System.out.println* desde JavaScript para imprimir mensajes en la consola Java.

En JavaScript, los *packages* Java y las clases son propiedades del objeto *Packages*. Usaremos la sintaxis Java para hacer referencia a objetos Java en JavaScript, con el nombre del objeto *Packages* incluido opcionalmente:

```
[Packages.].nombrePackage.nombreClase.nombre
    Metodo
```

Se puede incluir de forma opcional el nombre *Packages* para los *packages Netscape, java* y *sun*, siendo netscape, java y sun, en el código JavaScript, alias de Packages.netscape, Packages.java y Packages.sun.

El resto incluye el nombre *Packages.*

También usaremos la sintaxis Java para acceder a variables y métodos en una clase. Por ejemplo, las siguientes líneas de código JavaScript producirán el efecto de escribir en la consola Java:

```
Java.lang.System.out.println("Hola, que tal");
```

Podremos incluso usar constructores Java en JavaScript. El siguiente ejemplo creará un objeto Java (*Date*) e imprimirá su contenido en la consola Java:

```
var unObjeto = new java.util.Date()
System.out.println(unObjeto)
```

• Controlando applets desde JavaScript

JavaScript puede acceder a todas las variables públicas, propiedades y métodos de un applet Java. Por ejemplo, podríamos

utilizar botones en un formulario para arrancar y detener un applet en la página donde se encuentre.

A cada applet en un documento se le hace referencia en JavaScript con *document.nombreApplet*, siendo *nombreApplet* el valor del atributo *NAME* en el TAG <APPLET> correspondiente.

El array *applets* contiene todos los applets en una página, así que podremos referirnos a los elementos de este array con el nombre del applet correspondiente o con el número de la posición que el applet tiene en el array (como hemos visto en números anteriores, empieza en cero y va aumentando desde el principio de la página hacia abajo).

Como ejemplo vamos a crear un applet que imprima un mensaje en pantalla:

```
import java.applet.Applet;
import java.awt.Graphics;
public class simpleMensaje extends Applet {
  public void paint(Graphics g) {
    g.drawString("Esto es un Applet", 100, 40);
    }
  }
```

Para conseguir que aparezca en nuestra página usaríamos el siguiente código HTML:

```
<APPLET CODE="simpleMensaje.class"
        NAME="simpleMensaje"
        WIDTH=125 HEIGHT=50>
</APPLET>
```

Desde JavaScript podríamos hacer referencia a este applet de las siguientes maneras:

- document.simpleMensaje
- document.applets ["simpleMensaje"]
- document.applets[0]

Esta última manera supone que *simpleMensaje* es el primer applet dentro de nuestro documento. El array *applet* tiene una propiedad que nos permitirá conocer el número de elementos que contiene (esto es, el número de applets dentro del documento): *length*

JavaScript puede acceder a todas las variables públicas, propiedades y métodos de un applet Java

Vamos a ver a continuación un par de ejemplos más o menos complejos de comunicación entre JavaScript y Java.

Hay que modificar el applet *simpleMensaje* e incluir los estos cambios:

- El método *init* para declarar e inicializar un string llamado *elqueImprime*.
- Definir un método, al que llamaremos *elqueCambia*, que recibirá un String como argumento, se lo asignará a *elqueImprime* y llamará al método *repaint*.

El código sería:

```
import java.applet.Applet;
import java.awt.Graphics;

public class simpleMensaje extends Applet {
  String myString;

  public void init() {
    elqueImprime = new String("Esto es un
                                Applet");
  }

  public void paint(Graphics g) {
    g.drawString(elqueImprime, 25, 20);
  }

  public void elqueCambia(String aString) {
    elqueImprime = aString;
    repaint();
  }
}
```

Esto provoca que el mensaje que se imprima pueda ser modificado desde JavaScript. Cambiaremos el código HTML para ver cómo se haría:

```
<APPLET CODE="simpleMensaje.class"
        NAME="simpleMensaje"
        WIDTH=125 HEIGHT=50>
</APPLET>

<FORM NAME="form1">
<INPUT TYPE="text" SIZE="20"
        NAME="acambiar"
        VALUE="879000000">
<INPUT TYPE="button"
        VALUE="Cambia Mensaje"
    onClick=
    "document.simpleMensaje.elqueCambia
        (document.form1.acambiar.value)">
</FORM>
```

Ahora ya tendremos un form que consistirá en un botón que, al ser pulsado, cambiará el mensaje que contiene el applet, incluyendo el que nosotros hemos introducido dentro del Text Box llamado *acambiar*.

Veremos a continuación un ejemplo más complejo. Seguramente todo el mundo conocerá esos applets que realizan una cuenta atrás. Pues bien, mi primer applet fue una cuenta atrás y, como es bastante sencillo, lo vamos a utilizar como ejemplo.

El código Java del applet no está optimizado, es seguro que habrá un millón de maneras de hacerlo mejor; ruego disculpas por anticipado a los puristas de la programación.

El parámetro a pasar al applet es la fecha desde la que empezar a hacer la cuenta atrás. Esta fecha se le indica al applet por un mecanismo bastante complicado, baste decir que las cero horas del día 1 de Enero de 1998 se representa con el número 883609200. Si alguien tiene el tiempo y las ganas de hacer un algoritmo para convertir las fechas Java a algo humanamente entendible, por favor, que me lo envíe.

Vemos en el código fuente del applet (*cuentatras.java*) que las modificaciones hechas son muy parecidas a las del ejemplo anterior, simplemente incluimos la función *elqueCambia* para modificar la variable *tiem*.

Los cambios a realizar en el fichero HTML también son mínimos:

```
<APPLET code="cuentatras.class"
      name="cuentatras"
      width=480
      height=27>

<PARAM name="tiempo"
       value="870000000">
</applet>

<FORM NAME="form1">

<INPUT TYPE="text" SIZE="20"
      NAME="acambiar">

<INPUT TYPE="button"
       VALUE="Cambia Mensaje"
onClick= "document.cuentatras.elqueCambia
(document.form1.acambiar.value)">

</FORM>
```

Comunicación de Java a JavaScript

Aparte de acceder a nuestros applets desde JavaScript, también podremos realizar la operación inversa, esto es, acceder a los métodos, propiedades y estructuras de datos de JavaScript desde un Applet, solo tendremos que importar el *package* de Netscape llamado *javascript*:

```
import netscape.javascript.*
```

Este *package* define la clase *JSObject* y la excepción *JSException*. Para que podamos acceder desde un applet al código JavaScript en una página HTML, tendremos que añadir el atributo MAYSCRIPT al tag <APPLET> correspondiente. De esta manera, un applet Java no podrá acceder al código JavaScript de una página HTML sin que sea descubierto por el autor de la misma. En el caso de que un applet intentara acceder al código JavaScript desde un tag <APPLET> que no contiene el atributo MAYSCRIPT se generará una excepción.

Tabla 1

```
import java.applet.Applet;
import java.awt.*;
import java.util.Date;
public class cuentatras extends
Applet implements Runnable{
      String i;
      boolean pp=true;
      boolean suspende=false;
      String coje=null;
      Integer p;
      long tiem;
      Thread killme=null;
      public String getAppletInfo() {
                  return "cuentatras by Alejandro Reyero. axl@las.es";
      }
      public void init() {
                  setBackground(Color.black);
                  resize(480,27);
                  setFont(new Font("TimesRoman", Font.BOLD,24));
                  coje =getParameter("tiempo");
                  if (coje == null){
                     p = Integer.valueOf("883609200");
                     tiem=883609200; }
                  else{
                              p = Integer.valueOf(coje);
                              tiem = p.longValue(); }
                              setLayout(null);
                              addNotify();
      }
      public void start() {
         if (killme==null) {
            killme = new Thread(this);
            killme.start();
         }
      }

      public void stop() {
         killme=null;
      }

      public void run() {
         while (killme!=null) {
            try {
               Thread.sleep(900);
            } catch (InterruptedException e){}
            repaint();
         }
         killme=null;
      }
     public boolean mouseDown(java.awt.Event evt, int x, int y) {
         if (suspende) {
                  killme.resume();
         }
         else {
            killme.suspend();
         }
         suspende = !suspende;
         return true;
         }
public void paint(Graphics g) {
                  g.setColor(Color.red);
                  long testeo=0;
                  i="lelelelel";
```

Tabla 1. Continuación:

```
        while(pp==true){
                Date d1=new Date();
                long hace=d1.getTime();
                long haceseg=hace/1000;
                long pone=tiem-haceseg;
              long segundos=pone%60;
                long temp=pone/60;
                long minutos=temp%60;
                long temp2=temp/60;
                long horas=temp2%24;
                long dias=temp2/24;
                if (testeo!=pone){
                        g.setColor(Color.black);
                g.clearRect(0,0,480,27);
                g.setColor(Color.red);
                testeo=pone;
                if (dias!=1){
                        i= (String.valueOf(dias).toString()+
                                " dias, ");
                } else {
                        i= (String.valueOf(dias).toString()+
                                " dia, ");
                }
                if (horas!=1){
                        i=i+(String.valueOf(horas).toString()+
                                " horas, ");
                 }else {
                        i=i+(String.valueOf(horas).toString()+
                        " hora, ");
                         }
                if (minutos!=1){
                         if (segundos==0){
                                minutos=minutos-1;
                                segundos=60; }
                i=i+(String.valueOf(minutos).toString()+
                        " minutos y ");
                          }
                else {
                        i=i+(String.valueOf(minutos).toString()+
                        " minuto y ");
                }
                if (segundos!=1){
                        i=i+(String.valueOf(segundos).toString()+
                        " segundos ");
                 } else {
                   i=i+(String.valueOf(segundos).toString()+
                        " segundo ");
                          }
                if (haceseg >= tiem){
                        pp=false;
                   }
                 g.drawString(i,5,20);
                }
        }
        g.clearRect(0,0,480,27);
        g.drawString("Comentarios: ax1@las.es",0,20);
    }
    public void elqueCambia (String aString) {
        p = Integer.valueOf(aString);
        tiem = p.longValue();
        repaint();
    }
}
```

Para que un applet pueda acceder a código JavaScript tendrá que contener el atributo MAYSCRIPT en el tag <APPLET> correspondiente

En el applet, antes de que sea posible realizar el acceso a código JavaScript, tendremos que definir un *handle* (manejador) para la ventana de Navigator. Usaremos, para poder conseguir este acceso, el método *getWindow* que se encuentra en el interior de la clase *netscape.javascript.JSObject*. De esta manera, será posible definir el *handle*, pasándole el objeto applet.

Por ejemplo, si lo que se quiere es asignar un *handle* determinado a la variable *JSObject* llamada *ventana*, entonces lo que es necesario hacer es lo que se muestra a continuación:

```
public class nuestroApplet extends Applet {
  public void init() {
    JSObject ventana = JSObject.getWindow(this);
  }
}
```

Para poder acceder a las distintas propiedades y a los objetos que proporciona JavaScript, utilizaremos el método llamado *getMember*, dentro de la clase *netscape.javascript.JSObject*. Por lo tanto, utilizaremos *getWindow* para realizar la definición del *handle* para la ventana JavaScript, así como el método *getMember* para acceder a los objetos de que dispone JavaScript.

Por ejemplo, el código Java que mostramos a continuación nos va a permitir el acceso al objeto d*ocument.esunForm* a través de la variable *elForm*:

```
public void init() {
  ventana = JSObject.getWindow(this);
  JSObject doc = (JSObject)
       ventana.getMember("document");
  JSObject elForm = (JSObject)
       doc.getMember("esunForm");
}
```

Como podemos observar en este ejemplo, los objetos JavaScript aparecen como si fueran instancias de la clase conocida como *netscape.javascript.JSObject* en el lenguaje Java.

En las líneas que siguen a continuación, vamos a ampliar el ejemplo práctico que hemos desarrollado en el listado anterior. Ahora vamos a suponer que el objeto *document.esunForm.amplia* es un checkbox. Mediante el siguiente código Java se nos permitirá acceder a su propiedad *checked*:

```
public void init() {
  ventana = JSObject.getWindow(this);
  JSObject doc = (JSObject)
       ventana.getMember("document");
  JSObject elForm = (JSObject)
       doc.getMember("esunForm");
  JSObject check = (JSObject)
       elForm.getMember("amplia");
  Boolean chequea = (Boolean)
       check.getMember("checked");
}
```

• Utilizando métodos JavaScript desde Java

El método *eval* de la clase *netscape.javascript.JSObject* nos permite evaluar expresiones JavaScript de manera bastante sencilla, utilizando la siguiente sintaxis:

```
JSObject.getWindow().eval("expresion")
```

Como buen ejemplo de todo ello, veremos el siguiente código escrito en el lenguaje Java que utiliza la función *eval* para realizar la llamada al método JavaScript *confirm* cuando se produce cualquier tipo de evento (en este caso, se trataría de *mouseDown*, esto es, de la pulsación del botón del ratón).

```
public void init() {
  JSObject ventana = JSObject.getWindow(this);
}
public boolean mouseUp(Event e, int x, int y) {
  ventana.eval("confirm(\"Solo un Ejemplo.\");");
  return true;
}
```

Por otra parte, es posible llamar a los distintos métodos de que dispone JavaScript por medio del método *call*. En efecto, éste se utiliza para llamar a los métodos que tiene JavaScript desde un applet en el momento en que sea necesario pasar los objetos del lenguaje Java como argumentos:

```
JSObject.call(nombreMetodo,
          array_de_argumentos)
```

También podremos realizar la llamada de todas funciones de JavaScript que hayan sido creadas por nosotros mismos desde un applet.

En la práctica, supongamos, por ejemplo, que la función que mostramos a continuación se encuentra dentro de la etiqueta <HEAD> que encabeza nuestra página:

```
SCRIPT>
function miFuncion() {
  alert("Made by Solo Programadores")
}
</SCRIPT>
```

La que aquí presentamos es, como se puede ver, una función bastante simple que únicamente se dedica a abrir una ventana de alerta. Para poder hacer una llamada a esta función desde nuestro propio applet, tendremos que utilizar el método *call* tal y como se muestra en el siguiente listado:

```
public void init() {
  JSObject ventana = JSObject.getWindow(this)
  String argumentos[] = {""}
  ventana.call("miFuncion", argumentos)
}
```

Como se puede ver a través de este ejemplo, será necesario declarar el array *argumentos* en cualquier circunstancia, aunque éste se encuentre de hecho vacío (esto es, aun cuando no contenga argumentos de ningún tipo).

Conclusiones

En este artículo hemos aprendido a utilizar *LiveConnect* para comunicar nuestros applets con JavaScript y viceversa, lo que abre un nuevo horizonte de interactividad en nuestras páginas WEB.

LiveConnect también permite manejar plug-ins Java desde nuestras páginas. Una vez haya terminado la serie sobre programación JavaScript, aprenderemos a programar (y manejar) nuestros propios Plug-Ins.

Bibliografía

- "JavaScript: Una chispa de vida WEB", por Fernando J. Echevarrieta. Sólo Programadores número 30.
- "El Modelo de Objetos de JavaScript", por Alejandro M. Reyero. Sólo Programadores número 32.
- "JavaScript: Frames y Windows", por Alejandro M. Reyero. Sólo Programadores número 33.

Contactar con el autor

Si el lector quiere hacer llegar sus opiniones, preguntas, ejemplos, o, simplemente, decir "hola" al autor:

E-Mail: axl@las.es

WEB: http://www1.las.es/~axl

Criptografía de clave pública. RSA.

José Antonio Mendoza

Como no podía ser menos, este nuevo tipo de criptografía está basado en la teoría matemática de las funciones de un sólo sentido. Estas funciones tienen la propiedad de que la realización del cálculo de un valor según dicha función es fácil, pero resulta imposible computacionalmente realizar el cálculo del valor inverso si no se dispone de información privilegiada. Es decir, es fácil hallar f(a) = b, pero es imposible dado b hallar un a que satisfaga la ecuación anterior, $f^{-1}(b)=a$. Será muy fácil hallar **a** dado **b** si disponemos de determinada información, obviamente esa información será la contraseña.

Esta nueva criptografía apareció como un intento de mejorar la tradicional en dos de sus debilidades. En primer lugar, la distribución de claves; ya que emisor y receptor estaban condenados a mantener la misma clave, estos debían intercambiarla lógicamente de manera segura. Además, dicha clave se encuentra en dos sitios distintos, duplicándose la posibilidad de ser descubierta, con la particularidad de hacerse ingobernable el mantener una base de datos de claves cuando el número de interlocutores se hace un poco grande. Imaginemos una clave distinta para cada una de las personas con las que nos queramos comunicar.

El otro problema tenía relación con la autentificación del documento enviado. La criptografía tradicional aseguraba la autenticidad de un documento gracias a la legibilidad del texto una vez descifrado, pero ¿y si el mensaje a descifrar no tiene la regularidad de un texto, sino que son datos?, ¿quién nos garantiza que dicho mensaje no ha sido enviado por un impostor?

Cifradores de clave pública

Esta teoría de cifradores de clave pública, se debe a dos investigadores de la universidad de Stanford Diffie y Hellman, que en 1976 presentaron las ideas principales de la nueva rama de la criptografía, pero sin demostrar la existencia de métodos concretos. Sólo sentaron las bases de la nueva teoría criptográfica.

Éstas, de forma resumida, son las siguientes :

1.- Para un interlocutor es fácil generar el par clave pública y clave privada.
2.- Es fácil generar el mensaje cifrado: C=Eku[M].
3.- Es fácil para el receptor computar M=Ekr[C].
4.- Es computacionalmente inalcanzable dada la clave pública ku averiguar la clave privada kr.
5.- Es computacionalmente inalcanzable, conocido el mensaje cifrado C y la clave pública, ku recuperar M.
6.- Las funciones de cifrar y descifrar son aplicables en cualquier orden: M=Eku[Dkr[M]].

En el artículo anterior vimos la criptografía tradicional, caracterizada fundamentalmente por la utilización de la misma contraseña por parte de emisor y receptor para cifrar y descifrar el mensaje. A este tipo de cifrado se le denomina simétrico. El cifrado asimétrico, por el contrario, utiliza claves distintas en emisor y receptor.

Es el momento de definir las funciones que van a intervenir en todo este proceso. En Inglés son llamadas *trapdoor one-way function*, *trapdoor* literalmente significa escotilla, trampa. En castellano podría traducirse por funciones de un solo sentido con atajo, no queda muy académico pero resulta ilustrativo de las propiedades de dichas funciones. Éstas se pueden resumir diciendo que son funciones de ida, con vuelta muy dificultosa si no se conoce un atajo por el que volver de forma fácil. Matemáticamente quedaría así :

- $Y = f_k(X)$ fácil si k y X son conocidos.
- $X = f_k^{-1}(Y)$ fácil si k e Y son conocidos.
- $X = f_k^{-1}(Y)$ impracticable si Y es conocida pero no k.

Un ejemplo de función de un solo sentido sería por ejemplo un polinomio de grado 6 con unos cuantos términos, $y = x^6 + 10x^5 + 7x^3 + 2x^2 + 23$, dado $x = 23$, es fácilmente calculable el valor de y, basta sustituir el valor de x en el polinomio y ya está. Pero, dado y, no es trivial encontrar un valor de x que satisfaga la igualdad.

Por lo tanto nuestro problema será encontrar dicha función que cumpla con las propiedades anteriormente exigidas. Pero antes veamos los modelos de comunicaciones para los métodos de criptografía de clave pública y unas ideas que centren los aspectos fundamentales.

Como hemos dicho antes, los cifradores simétricos disponen de la misma clave en emisión y recepción, mientras que los cifradores asimétricos usan distinta clave en emisión y recepción. Por lo tanto cada interlocutor de la comunicación dispondrá de su clave. Dicha clave estará compuesta por una parte privada que se mantendrá secreta y otra parte que se hará pública. Según se emplee una u otra se obtendrán diferentes propiedades en la comunicación.

Modelos de comunicación

Como hemos dicho en el apartado anterior, al disponer cada interlocutor de una clave distinta y al estar formada ésta por dos partes, una pública y otra privada, se llegará a tres modelos de comunicación distintos, según se usen las claves de una manera u otra. La parte pública de la clave se difundirá entre los posibles interlocutores que nos quieran mandar mensajes, lógicamente la parte privada la mantendremos secreta. Los tres modelos son los siguientes :

1.- **Confidencialidad.** A quiere enviar un mensaje a B de forma secreta. Para eso A encriptará el mensaje con la clave pública de B, K_{UB}, que previamente B ha difundido o puesto en un directorio público. Cuando a B le llegue el mensaje, lo descifrará con su clave privada. Nadie salvo A y B conocen el contenido del mensaje, hemos conseguido una comunicación secreta. Solo queda un pequeño problema: B no sabe realmente si ha sido A el verdadero emisor del mensaje. Puede haber sido un impostor, pues la clave pública de B puede ser conocida por mucha gente. Ver figura 1 para este modelo.

2.- **Autenticidad.** A quiere enviar un mensaje a B auténtico, es decir, que sólo A lo haya podido haber emitido. Para eso difundirá su clave pública o la pondrá en un directorio público. De esta manera B estará en posesión de la clave pública de A. Cuando A quiera enviar un mensaje a B cifrará el mensaje con su clave privada y lo enviará a B, éste al recibirlo lo descifrará con la clave pública de A. Lógicamente B está seguro de que el mensaje es de A pues sólo A lo ha podido emitir, al disponer sólo él de su clave privada. Como era de prever, dicho mensaje no será secreto pues la clave pública de A es la que ha servido para descifrarlo, estando ésta a disposición de cualquiera. Hemos conseguido autentificar el mensaje, pero hemos perdido la privacidad de la comunicación. Ver figura 2 para este modelo.

3.- **Autenticidad y confidencialidad.** Para poder conseguir ambas necesidades, que el emisor sea auténtico y no un impostor, y para que nuestra comunicación sea secreta y no pública, tendremos que realizar un doble encriptado. Primero A cifrará con su clave privada, de esta manera se autentifica, después cifrará con la clave pública de B, de esa manera el mensaje permanecerá secreto salvo para A y B. En recepción se harán las operaciones inversas. Para este modelo ver figura 3.

Figura 1. Confidencialidad.

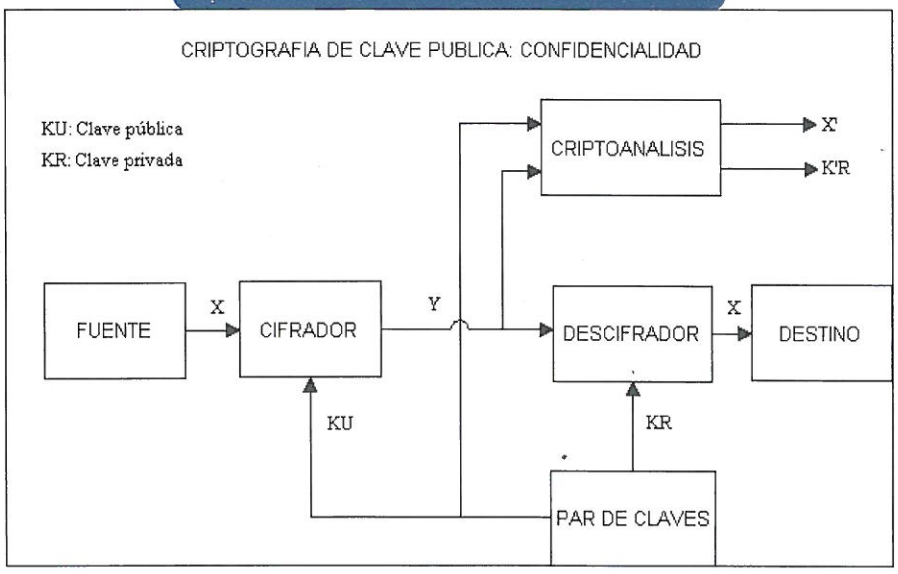
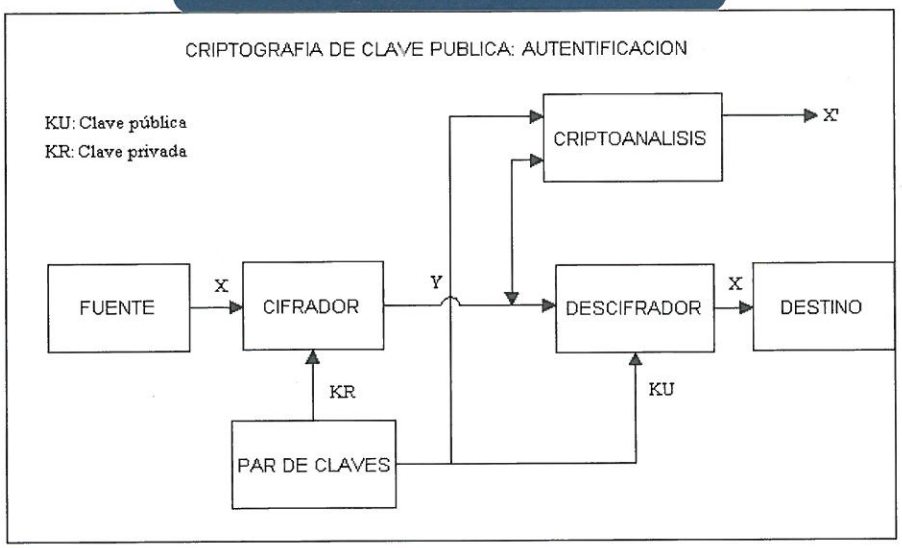


Figura 2. Autentificación.



Algoritmo de la mochila

Hemos visto las bases de esta nueva manera de cifrar las comunicaciones. Para empezar a ver cómo funcionan este tipo de algoritmos y como primera aproximación, comenzaremos por el famoso acertijo de la mochila, propuesto por Merkle en 1978. Se formula como sigue: dado el peso de una mochila y el peso de todos los objetos que pueda contener, adivinar la combinación de ellos que contiene. Si sólo tenemos dos objetos posibles, o está uno o está otro o están los dos, como sabemos el peso de la mochila, es obvio adivinar el contenido, siempre y cuando pesen distinto. Si fuera un número considerable (cien o doscientos) la cosa ya no sería tan fácil.

Aparentemente el anterior problema nada parece que tenga que ver con la criptografía. Sin embargo, en este ejemplo se encuentra una buena base para captar las ideas de la criptografía de clave pública. Veamos qué relación tiene.

Formemos el vector A= (a_1,a_2,a_3,....., a_n), constituido por los pesos de los objetos que pueda contener nuestra mochila, y el vector de datos X=(x_1,x_2,x_3,....., x_n), donde x_i , un elemento cualquiera del vector, será uno o cero, es decir, si está dentro de la mochila uno y si no cero. El peso de la mochila será $S=\hat{A}^{n}_{i=1}\, a_i\, x_i$. El problema de la mochila consiste en averiguar X, qué objetos están en la mochila, conocido S, el peso de la misma y A, el vector de pesos de todos los objetos. Antes de describir cómo resolver nuestro problema, tenemos que hacer una restricción a nuestro vector de pesos.

El algoritmo RSA libra a la criptografía del uso de una misma clave en emisor y receptor

Los valores del vector A deben ser elegidos de tal manera que dos combinaciones de elementos no den la misma suma, pues no podríamos saber qué combinación de X la ha generado, al haber dos o más, no seríamos capaces de conocer cuál de ellas es la verdadera.

Esto se evita haciendo que A sea incrementalmente lineal, es decir $a_1+,....,+a_{i-1}+a_i < a_{i+1}$; de esta manera, cuando introduzcamos un objeto en la mochila su peso lo delatará. Es como si en la mochila hubiera dos objetos: el más pesado de todos los presentes y la suma de los restantes, que nunca pesará más que el objeto más pesado. El proceso de cálculo de X a partir de S y A es trivial. Para calcularlo procedemos de la siguiente manera. Si S es mayor que a_n, entonces S contiene a a_n por lo tanto x_n =1, restaríamos a S a_n. En caso de ser menor no restaríamos nada y procederíamos de la misma manera con a_{n-1}.

De esta manera hemos conseguido hallar, de una manera fácil, el valor de X, a partir del mensaje enviado S y del vector A. El problema estriba en cómo conseguir que sólo el receptor sea capaz de descifrarlo. Merkle propuso lo siguiente: convirtamos nuestro vector incrementalmente lineal en otro que no lo sea, ya no dispondríamos del algoritmo anterior para hallar fácilmente X. Bastan dos enteros r y s para conseguirlo, siempre y cuando cumplan :

1.- $r > \hat{A}^{n}_{i=1}\, a_i$

2.- r y s son relativamente primos.

Ahora nuestro nuevo vector A' será : A'=rA mod s, es decir, los elementos de A' vendrán dados por la operación módulo s al producto de r por cada elemento de A, A'= (ra_1 mod s, ra_2 mod s ,....., ra_n mod s). La operación módulo consiste en calcular el resto de la división de dos enteros, en este caso $r*a_i$ mod s. S se calcularía igual que antes pero con el nuevo vector A'. Éste ya no será incrementalmente lineal, dificultando así el cálculo de X. Pero a primera vista podría parecer que ahora tampoco nosotros podemos utilizar dicho algoritmo. Sin embargo disponemos de r y s y con dichos valores podremos recuperar el vector X.

En primer lugar hay que calcular el valor de r^{-1}, el inverso de r. Es decir $r^{-1}\, r = 1$ mod s. dado un número entero su inverso en aritmética modulo s se puede calcular mediante el algoritmo de Euclides, el cual veremos más adelante.

Ahora con r^{-1}, X se calcula de forma trivial. Se procede haciendo el producto Sr^{-1} mod s, si es mayor que a_n , $x_n = 1$, que es menor $x_n = 0$. Ahora a Sr^{-1} mod s si es mayor que a_n se le resta a_n, si no, no se le resta nada y se procede a comparar con a_{n-1}, procediéndose de la misma manera hasta llegar a a_1.

Identifiquemos ahora los datos del ejemplo anterior con nuestro modelo de comunicaciones. Nuestro mensaje en claro: X, la clave pública: A', el algoritmo de cifrado: $\hat{A}^{n}_{i=1}\ a'_i\ x_i$, S el mensaje cifrado, como clave privada: r,s,A, y por último el algoritmo de descifrado, el explicado anteriormente de comparación y si procede resta.

Finalmente veamos un ejemplo: empezamos dando valores al vector A que, como sabemos, pertenece a la clave privada del receptor y debe ser incrementalmente lineal.

A=(1, 3, 9, 15 , 30 , 61 , 142).

Ahora nos toca elegir r y s, también pertenecientes a la clave privada del receptor:

1+3+9+15+30+61+142=261,

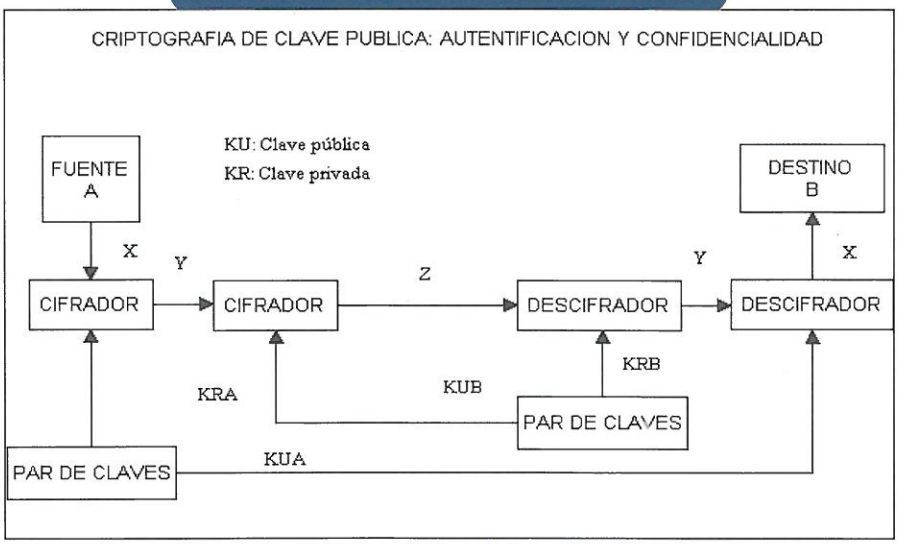
busquemos un número primo mayor que 261, por ejemplo 263, y un número relativamente primo con éste, es decir mcd(263,r)=1, por ejemplo otro primo 181; ya tenemos la clave privada del receptor:

{A(1,3,9,15,30,61,142) , r=181 , s=263 }.

Calculemos ahora r^{-1} , $181 * r^{-1} = 1$ mod 263, mediante Euclides, $r^{-1} = 93$, se comprueba fácilmente que $93*181=1$ mod 263.

El siguiente paso será calcular A', la clave pública. Haciendo los cálculos según $a'_i = r a_i$ mod s, llegamos a A=(181,17,51,85,170,258,191), que no es incrementalmente lineal. Si queremos transmitir el vector X(1,0,0,1,1,0,1), calculamos:

Figura 3. Confidencialidad y Autentificación.



S= 1*181+0*17+0*51+1*85+1 *170+0*258+1*191=627,

éste será nuestro mensaje cifrado, que transmitir por el canal público.

El receptor, si no hay errores de transmisión, recibirá 627, que deberá proceder a su descifrado de la siguiente manera.

627*93 mod 263 = 188
188 >=142 ; SI -> x7 =1 ; 188-142=46
46 >= 61 ; NO -> x6 =0 ; 46-0 = 46
46>= 30 ; SI -> x5 =1 ; 46-30=16
16>=15 ; SI -> x4 =1 ;16-15=1
1>=9 ; NO -> x3 =0 ;1-0=1
1>=3 ; NO -> x2 =0 ;1-0=1
1>=1 ; SI -> x1 =1 ;1-1=0

Vemos que hemos recobrado el vector original, sólo a partir de S y nuestra clave privada. Merkle retó a que alguien intentara romper este algoritmo, partiendo sólo de S y A', y puso como recompensa 100 dólares. A los cuatro años Adi Shamir lo logró y la cobró. Merke mejoró el algoritmo, a base de cifrados encadenados con diferentes parejas (r,s), y puso como recompensa 1000 dólares a quien lograra romperlo. Su desafío sólo duró dos años, llegándose a la conclusión de que los algoritmos basados en la mochila no son seguros para la criptografía de clave pública.

Como ya comentamos en el artículo anterior, un método se considera seguro hasta que alguien demuestra lo contrario.

Antes de empezar con el método de criptografía pública RSA (Rivest, Shamir, Adleman), tenemos que ver el algoritmo de Euclides y unos cuantos teoremas, relativos a los números primos.

Euclides y los números primos

Desde la antigüedad, los números primos han sido objeto de amplios estudios debido a sus propiedades particulares. Recordemos el concepto de número primo: número primo es aquél que sólo es divisible por uno y por él mismo: 2,3,5,7,11,13,... Los que no lo son, están formados por el producto de números, que sí son primos: 6=3*2. Los primeros números primos son muy fáciles de calcular pero, según van aumentando, la cosa se complica llegando un momento en que no podemos asegurar con certeza que un número sea o no primo. ¿Cuánto tiempo le costaría a un superordenador comprobar si un número de mil cifras es primo o no?, seguramente una eternidad, aunque

los primeros ordenadores de válvulas comparados con los más rápidos de hoy en día, nos indican que la velocidad de proceso no es precisamente una constante ¿quién puede decir si dentro de no se sabe cuántos años dispondremos de ordenadores capaces de realizar semejantes cálculos? Aunque bastará con hacer los números tan grandes como para que los ordenadores del momento no puedan con ellos.

Aumentando el tamaño de la clave se aumenta la robustez del RSA

Empecemos por describir el algoritmo de Euclides. Recordemos que es el utilizado para calcular el inverso de un número, referido a otro número, según la operación módulo. Supongamos un número entero r y otro s, queremos calcular el inverso de r según aritmética módulo s, esto es: $r\ r^{-1}=1 \bmod s$. Primero computamos la serie: $x_0 = s$, $x_1 = r$, $x_{i+1} = x_{i-1} \bmod x_i$, hasta que $x_k = 0$, $x_{k-1} = \text{mcd}(r,s)$. Indicamos ahora cómo con estos cálculos obtendremos el inverso de r. Para cada x_i tenemos que calcular a_i y b_i, tal que se cumpla la expresión: $x_i = a_i x_0 + b_i x_1$, donde $a_{i+1} = a_{i-1} + y_i a_i$ y $b_{i+1} = b_{i-1} + y_i b_i$, e $y_i = x_{i-1}/x_i$. Entonces, si $x_{k-1} = 1$, el inverso de r será $r_{-1} = b_{k-1}$. Es posible que el valor anterior salga negativo, en ese caso le sumamos el valor de s. Como en el algoritmo anterior, lo veremos mejor con un ejemplo.

Tabla 1.

i	xi	ai	bi	yi
0	263	1	0	
1	181	0	1	1
2	82	1	-1	2
3	17	-2	3	4
4	14	9	-13	1
5	3	-11	16	4
6	2	53	-77	1
7	1	-64	93	

Vamos a calcular el inverso del apartado anterior, r = 181 y s = 263. Ver tabla 1. Vemos por tanto que $r^{-1} = 93$ y ya tenemos el inverso de 181 en aritmética 263.

Visto el algoritmo de Euclides, veamos ahora unas cuantas propiedades de los números primos, en las que se basa el algoritmo de cifrado público RSA, tan extendido en nuestros días.

Se define el número de Euler, ø(n), como el número de enteros positivos menores que n, que sean relativamente primos a n. Dos números son relativamente primos cuando su máximo común divisor es uno: mcd(a,b)=1. El número de Euler de un número primo, p es ø(p)=p-1, lógicamente el máximo común divisor de p con todos los números menores que él, es uno, por ser éste primo.

El Teorema de Euler se formula como sigue: dados a y n relativamente primos entonces se cumple: a^ø(n) = 1 mod n. Una manera más apropiada de enunciar el teorema, con vistas a la aplicación en el algoritmo RSA, es la siguiente: $m^{(p-1)(q-1)} = 1 \bmod n$ donde n=pq y 0<m<n, siendo p y q dos números primos, además mcd (m,n)=1, m y n relativamente primos.

RSA: Rivest, Shamir, Adleman

Los tres investigadores del MIT (*Massachusetts Institute of technology*) propusieron el método que lleva por nombre sus iniciales. Tenemos que aclarar ahora que este algoritmo de cifrado está patentado en los Estados Unidos y Suiza, países donde las autoridades reconocen el derecho de los investigadores a patentar algoritmos matemáticos, cosa que no pasa en la Comunidad Europea y Japón. También, por motivos estratégicos, el software que se base en estos métodos no es exportable fácilmente, ya que es considerado como un arma. La solución de dichas empresas ha sido crear filiales en otros países y desarrollar en ellos dicho software. No perdamos de vista la dimensión militar de todo este tipo de tecnologías, espionaje, manipulación de las comunicaciones, privacidad de las mismas etc. Una parte del PGP (*Pretty Good Privacy*), desarrollado por Phil Zimmermann en 1995, está basado en el RSA. PGP permite unas comunicaciones seguras en internet y existe una versión de libre distribución. Para más información consultad el artículo sobre el tema en Solo Programadores nº 32. Los estados no son partidarios de que los ciudadanos tengan comunicaciones seguras, al margen de la seguridad que pueda proporcionar el propio estado sobre las mismas, tutela judicial etc.

Tras este paréntesis sobre las implicaciones políticas, militares y estratégicas de las tecnologías de cifrado de datos, adentrémonos en los fundamentos de funcionamiento del RSA.

RSA no sólo consigue privacidad, sino también autentificación

Lo fundamental del RSA es que está basado en la dificultad que tienen los ordenadores para encontrar números primos cuando éstos son de un tamaño considerable. Los ataques de fuerza bruta se hacen imposibles, la capacidad computacional de los ordenadores actuales no puede con la avalancha de operaciones necesarias para desencriptar un mensaje cifrado sin la contraseña. Para darnos una idea de los órdenes de magnitud que manejamos, para factorizar un número, es decir, descomponerlo en sus factores pri-

mos, si éste es de 200 dígitos se necesitarían billones de años, suponiendo que nuestro ordenador ejecutara una instrucción cada microsegundo.

El RSA es un algoritmo de clave pública, lo cual quiere decir que el interlocutor que quiera recibir mensajes cifrados deberá hacer pública una determinada información, la cual será usada por el emisor para cifrar el mensaje.

Llamaremos Kua a la clave pública de A y a Kra a la clave privada de A. La composición de las claves sería la siguiente: la clave pública Kua={e,n} y la privada Kra={d,n}. El proceso para generar dichas claves es el que describimos a continuación. Primero elegimos dos números primos muy elevados, mayores de 10^{100}, p y q, n=pq , una vez que tenemos n, tenemos que elegir d siempre y cuando cumpla la siguiente condición:

$$mcd\ (\phi(n)\ ,d\ ) = 1\ ;\ 1<d<\ F(n);$$

sólo nos queda calcular e, $e = d^{-1} \bmod \phi(n)$. Podemos ver aquí una aplicación del algoritmo de Euclides, visto anteriormente.

Veamos ahora cómo se cifra y descifra con el RSA. Llamaremos C al mensaje cifrado y a M al texto en claro. Para cifrar, cogemos el mensaje en claro y hacemos la siguiente operación: $C= M^e \bmod n$. Para descifrar, realizamos la operación contraria: $M= C^d \bmod n$. Se debe cumplir que M<n.

La relación que guarda lo anterior con los teoremas de los números primos se encuentra en que basta sustituir la expresión de cifrar en la de descifrar: $M= M^{ed} \bmod n$; la relación que debe existir entre e,d y n para que dicha igualdad se satisfaga viene dada según el teorema de Euler que satisface $m^{(p-1)(q-1)} = 1 \bmod n$. De esta forma debería cumplirse: de= (p-1)(q-1) +1 que, por definición: $\phi(n) = \phi(pq)$ y como p y q son primos: $\phi(pq) = (p-1)(q-1)$, por lo tanto debe cumplirse que: $de = \phi(n)+1$, lo que es equivalente a: $de= 1 \bmod n$. Veamos un ejemplo para fijar ideas.

1.- Elijamos dos números primos, por ejemplo: p=5 , q= 17, n=5x17=85, $\phi(n)=(p-1)(q-1)=4x16=64$

2.- Ahora elijamos el valor de d, que debe ser relativamente primo con $\phi(n)$, es decir con 64, por ejemplo: 29, d=29.

3.- Debemos encontrar ahora el valor de e que, aplicando Euclides como las anteriores veces, llegamos a: $e=dd^{-1}=1 \bmod F(n)$, $29d^{-1}=1 \bmod 64$ y, resolviendo los cálculos: e= 53.

Ya tenemos la clave pública Ku{53,85} y la privada Kr{29,85}

Una vez hecha pública Ku, el emisor realiza con ella la siguiente operación:

$$C= M^e \bmod n$$

Suponiendo que el mensaje en claro que se quiere transmitir es M=45
$C=45^{53} \bmod 85= 10$, en recepción nos llegaría 10 y éste sería nuestro texto a descifrar:

$M= C^d \bmod n$, $M=10^{29} \bmod 85 =45$

Una vez explicado en qué consiste el RSA, cómo generar sus claves, pública y privada y cómo cifrar y descifrar, nos quedaría comentar que, además de cumplir lo necesario para que el algoritmo funcione, p y q deberían ser elegidos siguiendo los siguientes criterios, para darle más robustez:

1.- Diferir en pocos dígitos y ser superiores a 10^{100}.

2.- p-1 y q-1 deben contener un número primo como factor grande.

3.- El máximo común divisor de p-1 y q-1 debe ser pequeño.

El RSA tiene la limitación de no poder cifrar bloques grandes de datos y de necesitar muchas operaciones a la hora de cifrar y descifrar. Por último, la distribución de claves incluso aquí no es un asunto trivial. Como ejemplo, alguien podría suplantar la identidad de otra persona, diciéndolo y adjuntando una clave pública, y de esta forma los mensajes cifrados irían encaminados a un impostor.

Existe toda una teoría para la distribución de claves, para que no se produzcan falsas identidades y otros agujeros de seguridad, pero este tema no es el objeto del presente artículo.

Conclusiones

En el artículo hemos profundizado en los fundamentos de la criptografía de clave pública: los matemáticos, los computacionales y los operativos.

Hay que resaltar lo que es una constante en criptografía: no se puede asegurar al cien por cien que algo es seguro, aunque los científicos demuestran que estas técnicas lo son y que, si se ven amenazadas por la cada vez mayor capacidad de cálculo de los ordenadores y de muchos ordenadores trabajando conjuntamente, gracias a internet, basta aumentar el tamaño de las claves.

Todas estas tecnologías acaban de aparecer, son muy recientes y, a medida que transcurran los años y más y más científicos y matemáticos dediquen su tiempo y su esfuerzo, se conseguirá dominarlas. Existe el reto de calcular grandes números primos de una forma rápida y eficiente.

Bibliografía

- *Network and Internetwork Security*, William Stallings, editorial Prentice-Hall.
- *Computer Networks*, Andrew S. Tanenbaum, editorial Prentice-Hall.
- *Digital Communications*, Bernard Sklar, editorial Prentice-Hall.

AWS

C/ SERRANO JOVER, 3
28015 MADRID
TELEF.: 559 82 86

AMPLIACIONES

486 A PENTIUM
25.000 PTS

CD x4 A CD x16
8.500 PTS

DISCO DURO MAYOR DE 200 MB A 2 GB
22.000 PTS

PENTIUM 120 A 166
14.000 PTS

TODOS ESTOS PRECIOS INCLUYEN LA MANO DE OBRA

PORTATILES

COMPAQ LT - 5000

INTEL PENTIUM 75 MHZ
8 MB DE RAM
DISCO DURO DE 510 MB
MONITOR DSTN COLOR

154.000 PTS

olivetti ECHOS 100 D

INTEL PENTIUM 100 MHZ
8 MB DE RAM
DISCO DURO DE 810 MB
MONITOR DSTN COLOR

162.000 PTS

3 AÑOS DE GARANTIA EN PORTATILES
OFERTA VALIDA HASTA FIN DE EXISTENCIAS

INTERNET

FAX-MODEM 33.600 BPS

1 AÑO DE CONEXION CON TARIFA PLANA (TIEMPO ILIMITADO)

SERVIDOR CON UN ANCHO DE BANDA DE 34 MBS

15.500 PTS

PENTIUM 166

INTEL PENTIUM 166 MHZ
PLACA INTEL TRITON 512 CACHE
DISCO DURO 2 MB
32 MB EDO RAM
MONITOR 14 SVGA N.E. 0.28
T. VIDEO S3 TRIO 2 MB
CD X16
T. SONIDO 16 BITS

129.900 PTS

CON MMX 166 MHZ
+ 19.500 PTS

ESTOS PRECIOS NO INCLUYEN I.V.A. OFERTAS VALIDAS SALVO ERROR TIPOGRAFICO. LOS PRECIOS PUEDEN VARIAR SIN PREVIO AVISO.

¿TE ATREVES?

Desde **Sólo Programadores** estamos preparándonos para un torneo muy especial, de ordenador a ordenador, de programador a programador. El torneo de ajedrez está empezando a tomar forma y éstas son las bases, aunque también podéis informaros en AWS en la calle Serrano Jover, 3, de Madrid, o en el teléfono (91) 5 42 50 87. Para conocer cuál será el alcance de la participación, os pedimos que os inscribáis enviando vuestros datos a la dirección de AWS o a la dirección de e-mail: **solop@towercom.es**. El periodo de inscripción ya ha comenzado y terminará el 30 de noviembre, y la fecha límite para la recepción de los programas es el 15 de diciembre de 1997. La celebración del torneo será anunciada con suficiente antelación y podréis participar en persona o desde vuestra propia ciudad, ya que vuestro programa os representará y en todo momento estará presente un notario.

BASES DEL TORNEO

1. Se presentará un programa que juegue al ajedrez recibiendo la jugada del contrario y devolviendo la propia. El tiempo de la partida será medido por un juez y la partida se jugará físicamente en una mesa con un tablero siguiendo las instrucciones que cada ordenador dé; cada ordenador avisará de que ha elegido un movimiento con un pitido y mostrará su jugada en la pantalla de la forma: casilla a mover - casilla destino. La notación de la casilla vendrá dada por su columna y su fila. Para las columnas se utilizarán las de la A a la H siendo la columna A la primera columna de la izquierda, visto el tablero desde la posición de las blancas. Para las filas utilizaremos los números del 1 al 8 siendo la fila 1 la más cercana a nosotros (visto el tablero desde el lado de las blancas). Como ejemplo, el movimiento del peón de rey dos lugares hacia adelante sería: "E2 - E4". La jugada que hace el contrario se anotará en el ordenador de forma manual tecleando "E2-E4" <RETURN>.
2. El programa jugará al ajedrez según las normas de este juego, admitiendo todos sus movimientos.
3. El programa se ejecutara bajo Windows 95, MS Dos 6.22 y Linux, en un Pentium 166 con 16 MB de RAM.
4. El programa podrá jugar con las blancas o con las negras indistintamente.
5. Se debe entregar el código fuente, la versión del compilador utilizado y una breve documentación para su utilización.
6. Un programa podrá ser descalificado si realiza un movimiento no válido y esto deberá ser advertido por el contrario; por lo tanto, un movimiento será válido siempre que el otro programa no nos advierta de lo contrario. De la misma forma, si un programa nos advierte de un movimiento erróneo y éste no lo es, será descalificado.
7. Los jueces son los encargados de advertir las posiciones de tablas, el programa no advertirá de ello.
8. El programa debe detectar si ha ganado la partida y quedará descalificado ante cualquier detección errónea. No se avisará al contrario en el caso de que deje al rey al descubierto y ganará la partida el que antes mate al rey.
9. La metodología del concurso se reservará hasta que sea conocido el número de participantes (eliminatoria, puntos, etc.)
10. No se podrán utilizar en el programa bases de datos adicionales.
11. Los derechos del programa ganador pasarán a la propiedad de la empresa Tower Communications, SRL.
12. El premio consistirá en un Pentium PRO 200 con 64 MB RAM, VIRGE4 MB, CDX 16, disco duro 5 GB, monitor 17", teclado, ratón y disquetera.

CONCURSO DE PROGRAMACIÓN

SÓLO PROGRAMADORES Y AWS INFORMATICA
SERRANO JOVER 3, MADRID

MÁS DATOS EN EL PRÓXIMO NÚMERO

VISUAL BASIC INTERMEDIO

Las bases de datos en Visual Basic

Jordi Agost Moré

Hoy en día la mayoría de aplicaciones en Visual Basic poseen un cierto acceso a base de datos, veremos cómo operar con bases de datos casi sin programación, gracias a los objetos que nos ofrece Visual Basic.

En Visual Basic podemos realizar el acceso a las bases de datos con dos tipos de herramientas: los objetos de acceso de datos (*Data Access Object* DAO) y los controles de datos (o *Data Control*).

Los DAO nos permiten el acceso a bases de datos mediante la programación y no tienen interfaz de usuario (es decir no los encontraremos en la caja de herramientas). Entre ellos podemos destacar: los *Recordsets* (divididos en *Snapshots*, *Tables* y *Dynasets*), los *Database* y los *QueryDef*.

El control de datos permite llevar a cabo la creación de aplicaciones que visualicen y editen la información de diversos tipos de bases de datos, como podrían ser las siguientes: Microsoft Access, Microsoft Fox Pro, Btrieve, Dbase y Paradox.

La finalidad del control de datos es la implementación rápida de aplicaciones de bases de datos, sin apenas tener que escribir nada de código, no obstante llegaremos a un punto donde será más complicado continuar efectuando operaciones en las bases de datos con el control de datos que a través de los DAO (mediante la programación). De momento vamos a examinar los controles de datos. Para colocar un control de datos en un formulario debemos buscar en la caja de herramientas la correspondiente al control de datos, seleccionarla y a continuación dibujar el control sobre el formulario.

Los controles de datos en realidad sólo son un intermediario entre la base de datos y los controles vinculados a ellos, también llamados controles enlazados. Dichos controles nos permitirán el acceso a un determinado campo de la base de datos a través del control de datos. Si tenemos un control enlazado a un control de datos y éste se mueve al registro siguiente, entonces dicho control enlazado cambiará para mostrar los datos del campo del siguiente registro activo, al igual que todos los controles enlazados vinculados a nuestro control de datos, además, cuando se cambien los datos en un control enlazado y nos movamos a un registro diferente, los cambios se guardarán automáticamente en la base de datos. No obstante, hemos de tener en cuenta que los controles de datos sólo nos mostrarán un registro a la vez.

Para unir el control de datos a una base de datos lo haremos especificando la base de datos y la información a tratar mediante las siguientes propiedades del control de datos:

DatabaseName
Connect
RecordSource

La propiedad *DatabaseName* establece qué fichero de la base de datos queremos. La propiedad *Connect* contiene el valor para especificarle a VB el tipo de BD a la cual nos queremos conectar. Si lo desplegamos veremos una lista con los valo-

res de las posibles bases de datos candidatas, si no encontramos nuestra base de datos en la lista podremos conectar vía ODBC para hacer un acceso local o remoto a una BD. La propiedad *RecordSource* establecerá una sentencia SELECT o una tabla, de la cual extraeremos los datos que nos interesen. Recordemos en este punto que una base de datos es todo el conjunto de información que manejamos y que ésta puede estar subdividida en tablas. Por ejemplo, en una facturación tendríamos la base de datos de facturación y dentro podríamos tener las diferentes tablas de clientes, proveedores, artículos, stock, etc.

Tal y como hemos dicho, poniendo el nombre de una tabla accederíamos a ella, pero si queremos acceder a más de una tabla podemos utilizar la sentencia SELECT del lenguaje SQL (*Structured Query Language*). También podemos utilizar esta sentencia para realizar un filtrado de los campos que mostrar o de los diferentes registros en base a un determinado criterio.

Si ahora cogemos, por ejemplo, dos *TextBox* y las ligamos con el control de datos, poniendo su propiedad *DataSource* a Data1 (que es nuestro control de datos) y *DataField* con el campo que deseemos mostrar del conjunto posible, y acto seguido nos vamos moviendo por el control de datos, veremos cómo vamos cambiando de registro.

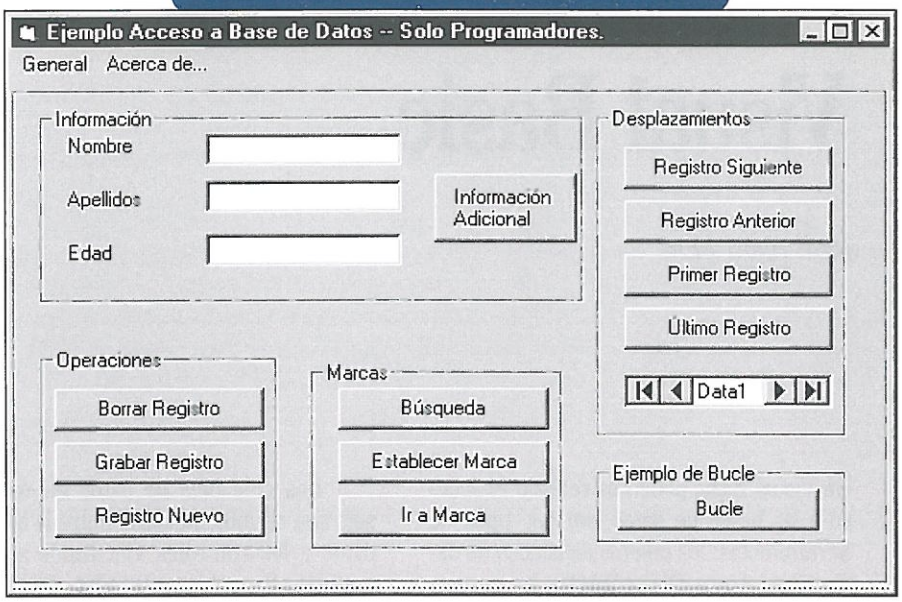
Los controles para cambiar de registro aparecen en la figura 1.

Podríamos obtener el mismo resultado con las siguientes sentencias, por ejemplo (ver código en los botones de ejemplo):

```
Data1.Recordset.MoveFirst
Data1.Recordset.MovePrevious
Data1.Recordset.MoveNext
Data1.Recordset.MoveLast
```

Con lo cual indicamos que del control de datos (Data1) y de su conjunto de registros (*Recordset*), vaya respectivamente al primer elemento, al anterior, al siguiente y al último.

FIGURA 1. Significado de los botones del control de datos.



De momento, en la *Text Box* sólo tenemos ligado un campo, pero puede ser que ocasionalmente necesitemos el valor del campo en una variable o internamente en algún asunto de programación, por ello podríamos poner (ver en el ejemplo el botón "Información Adicional") el siguiente código:

```
Data1.Recordset("Nombre").Value
```

o bien:

```
Data1.Recordset("Nombre")
```

De igual forma se actuaría para cualquier campo que deseemos obtener.

Puede ser que al conseguir el valor de un determinado campo nos encontremos con un mensaje del tipo: ... que nos indicará que el campo tiene un valor nulo, lo solucionamos con el siguiente código:

```
If IsNull(Data1.Recordset
    ("Nombre_del_campo").Value)
Then
    A$ = ""
else
    A$ = Data1.Recordset
    ("Nombre_del_campo ").Value
endif
```

con el código equivalente

```
A$=Iif(Data1.Recordset
("Nombre_del_campo").Value, Null, "")
```

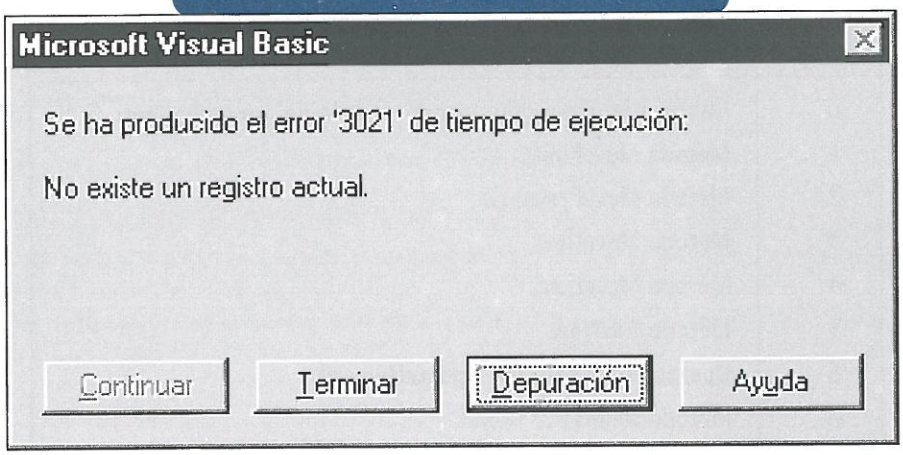
Movimiento a través de los registros

Dos propiedades interesantes del control de datos y su conjunto de registros (*Data1.Recordset*) son las *Eof/Bof* que nos indicarán si estamos en el inicio o al final del fichero. Por ejemplo, para ver todos los registros del conjunto (botón Bucle) podemos hacer:

```
Data1.Recordset.MoveFirst
While Not Data1.Recordset.EOF
MsgBox "Nombre y Apellidos: " &
Data1.Recordset.Fields("Nombre").Value & " " &
    Data1.Recordset.Fields("Apellidos").Value
 Data1.Recordset.MoveNext
Wend
```

Dichas propiedades las utilizaremos también para solucionar un error de este tipo cuando nos aparezca el mensaje que

FIGURA 2. Error de reposicionamiento en registros.



se muestra en la figura 2 en esta misma página:

El mensaje de error querrá decir que no tenemos un registro actual o que el registro actual no es válido, lo que sucederá, por ejemplo, si estamos en el último registro y ejecutamos un método del tipo *MoveNext*, con lo cual sobrepasamos el último registro. Para solucionar dicho error podemos poner el siguiente código:

```
Data1.Recordset.MoveNext
If Data1.Recordset.Eof Then
    Data1.Recordset.MoveLast
Endif
```

Otra propiedad útil puede ser *Bookmark*, que nos permitirá establecer una marca en un determinado registro. En el ejemplo hemos definido la variable Marca como de tipo carácter y global a nivel de formulario, luego para establecerla usamos el siguiente código:

```
Marca = Data1.Recordset.Bookmark
```

y para restablecer la marca (volver al registro marcado) usamos la instrucción:

```
Data1.Recordset.Bookmark = Marca
```

Para realizar una búsqueda de registros lo haremos mediante el método *FindFirst*, que nos buscará el primer registro que cumpla un cierto criterio. En el ejemplo (en el código del botón de búsqueda) se pone previamente una marca para volver al punto donde estábamos si no encontramos el registro. Sabremos si lo hemos encontrado o no consultando la propiedad *NoMatch* del *Recordset*, que adquirirá el valor de cierto (*True*) cuando no se haya encontrado, y de falso (*False*) cuando sí se encuentre. Por ejemplo:

```
Private Sub CommandBusqueda_Click()
Dim Edad As Byte
Dim Actual As String
' Establecemos una marca
Actual = Data1.Recordset.Bookmark
Edad = InputBox("Dime la edad a buscar: ",
      "Búsqueda")
Data1.Recordset.FindFirst "Edad = " & Edad
' si no se ha encontrado volvemos al punto anterior
If Data1.Recordset.NoMatch Then
 Data1.Recordset.Bookmark = Actual
End If
End Sub
```

Además, de igual forma que hemos utilizado el método *FindFirst* para buscar el primer registro que cumple un criterio, podríamos utilizar también los métodos *FindNext*, *FindPrevious* y *FindLast* para obtener el siguiente registro, el previo y el último que cumplen un determinado criterio.

Operaciones con registros

Para borrar un registro de la base de datos utilizaríamos los siguientes métodos:

```
Data1.Recordset.Delete
Data1.Recordset.Movenext
```

Para añadir un registro nuevo, el método será:

```
Data1.Recordset.AddNew
```

Dicho método nos crea un registro en blanco que además pasa a ser el nuevo registro activo. Acto seguido rellenamos los controles ligados con la información que deseemos y hacemos click en cualquier lugar del control de datos (automáticamente iremos a otro registro) para grabar.

Para modificar un registro lo único que tenemos que hacer es escribir direc-

FIGURA 3

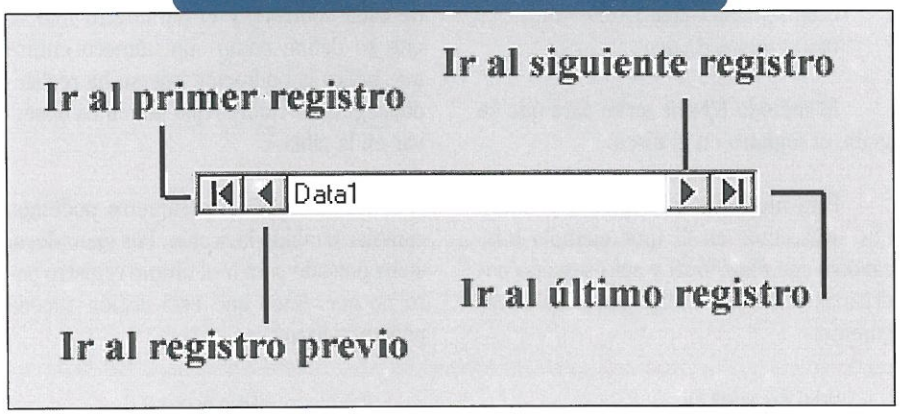


Tabla 1. Significado del parámetro *Action* del evento *Validate*.

	Constante	Valor	Descripción
fila1	vbDataActionCancel	0	Cancela la operación cuando se sale del procedimiento Sub.
fila2	vbDataActionMoveFirst	1	Método MoveFirst.
fila3	vbDataActionMovePrevious	2	Método MovePrevious.
fila4	vbDataActionMoveNext	3	Método MoveNext.
fila5	vbDataActionMoveLast	4	Método MoveLast.
fila6	vbDataActionAddNew	5	Método AddNew.
fila7	vbDataActionUpdate	6	Operación Update (no UpdateRecord).
fila8	vbDataActionDelete	7	Método Delete.
fila9	vbDataActionFind	8	Método Find.
fila10	vbDataActionBookmark	9	La propiedad Bookmark no ha sido definida.
fila11	vbDataActionClose	10	Método Close.
fila12	vbDataActionUnload	11	El formulario se va a descargar.

tamente sobre los controles ligados (las *Text Box*) y movernos a otro registro para que se grabe automáticamente.

Hasta ahora sabemos hacer nuevos registros, grabarlos y borrarlos. Ahora vamos a realizar las mismas operaciones pero mediante la programación. Las novedades aparecen cuando hemos de dar un registro de alta o hacer modificaciones en él. Por ejemplo, para dar un registro de alta podríamos utilizar el siguiente código:

```
Data1.Recordset.AddNew
Data1.Recordset("Nombre") = "Jordi"
Data1.Recordset("Apellidos") = "Casals"
Data1.Recordset("Telefono") = "111-
112233"
Data1.Recordset("Edad") = 26
Data1.Recordset.Update
```

El método *Update* sirve para que se grabe el registro en el disco.

Para modificar un registro, primero nos situaremos en él (por ejemplo buscándolo con *FindFirst*) y acto seguido podríamos tener un código parecido al siguiente:

```
Data1.Recordset.Edit
Data1.Recordset("Edad")=27
Data1.Recordset.Update
```

Como último punto destacaremos los eventos *Validate* y *Reposition* del control de datos.

El evento *Validate* se produce antes que un registro pase a ser el registro actual. Dicho suceso ocurre como paso previo a una operación *Update* y nos puede servir para validar los datos introducidos.

Validate tiene dos parámetros: *Save,* que contiene el valor de cierto (*True*) si algún control ligado al control de datos ha cambiado (podemos saber el control que es examinando la propiedad *Datachanged* de cada control), y el parámetro *Action,* que se define como un número entero que indica la operación que se ha realizado, según los valores que podemos observar en la tabla 1.

Modificando el parámetro podemos cambiar también la acción. Por ejemplo, si se ha pulsado para ir al último registro pero no queremos que esta acción suceda podemos hacer:

```
If Action = 4 Then Action = 0
```

en nuestro ejemplo, hemos controlado que edad sea un valor numérico:

```
If Trim(TextEdad.Text) <> "" And
Not (IsNum(TextEdad.Text)) Then
 MsgBox ("Edad no es un número")
 Action = 0
 Save = False
End If
```

Por último tenemos que decir que el suceso *Reposition* se produce después que el registro pase a ser el actual (éste es un buen momento para actualizar información de ciertos controles o para ocultar la que no estamos interesados en que se vea).

Hemos visto hasta el momento una serie de métodos que nos permiten el acceso a bases de datos y también hemos explicado cómo se efectúan diversas operaciones con ellas.

Aunque por ahora no hemos explotado la totalidad de la potencia que Visual Basic proporciona con las bases de datos, este artículo nos servirá como una buena base para que podamos empezar a hacer nuestros programas con acceso a bases de datos.

Seguridad en Novell Netware

Enrique Díaz Trobo

A muchos administradores de red, por no decir a todos, nos ha ocurrido que un usuario torpe nos ha borrado un directorio sobre el que teóricamente no tenía derechos. Rápidamente sospechamos que hubo trampa y pensamos que alguien le dio derechos sobre ese directorio. Y ese alguien no fue otro que nosotros mismos. Se ha producido un fallo en la seguridad. Situaciones como ésta son consecuencia de un descuido o una mala percepción acerca de cómo se combinan los derechos explícitos y los derechos heredados para dar lugar a los derechos efectivos.

El sistema de seguridad de Novell consiste básicamente en una jerarquía de usuarios que tienen unos determinados derechos sobre los objetos del árbol de directorios y sobre las propiedades de estos objetos, y también derechos que puede heredar de objetos padre o por equivalencia de seguridad. Alguno de estos derechos adquiridos puede verse limitado por el filtro de derechos heredados. A primera vista este sistema basado en los derechos, las propiedades y las herencias puede parecer complicado, sobre todo si tenemos en cuenta que también hay objetos que tienen derechos sobre otros objetos que a su vez han podido heredar derechos de un objeto padre. Quizá sería más fácil dar a cada usuario sus derechos y dar por terminado el problema. Si así fuera, el administrador debería darnos derechos explícitos sobre todos y cada uno de los elementos que vayamos a necesitar en nuestro trabajo en la red. Esto significaría que, por ejemplo, para poder trabajar en un directorio, el administrador tendría que darnos derecho sobre ese directorio y luego sobre todos los archivos de ese directorio, uno a uno. Gracias a la herencia bastará con que nos den derechos sobre el directorio, ya que adquiriremos esos mismos derechos para los archivos del directorio. Así pues, la herencia no es una complicación, sino una herramienta de primer orden a la hora de administrar una red Novell.

La jerarquía

En primer lugar veremos con quién nos estamos jugando los cuartos (la jerarquía administrativa y de usuarios), y más adelante entraremos a fondo con los derechos.

La herencia constituye una herramienta de primer orden a la hora de administrar una red Novell

Los usuarios de Novell se pueden categorizar en administradores de red,

La seguridad es crucial en cualquier sistema operativo de red. Si la administramos correctamente nos protegeremos tanto de las torpezas de los usuarios como de personas no autorizadas que nos roben o corrompan datos.

subadministradores, operadores y usuarios. El usuario *admin* se crea durante la instalación y tiene derechos de supervisor en la raíz del árbol NDS, por lo que puede hacer y deshacer con entera libertad en cualquier punto de la red. Los subadministradores son usuarios a los que se les otorga el derecho de supervisor a partir de una rama del árbol NDS, por ejemplo sobre una unidad organizativa. Por debajo de éstos suelen estar los operadores, usuarios a los que se les dan los permisos necesarios para administrar partes muy concretas de la red, como servidores, colas de impresión o grupos de usuarios.

Finalmente quedan los usuarios que no tienen ninguna responsabilidad en la red y se limitan a utilizarla. Toda esta 'burocracia' nos puede parecer excesiva, pero no debemos perder de vista que las versiones 4 de Novell están pensadas para administrar desde pequeñas redes locales hasta grandes redes mundiales.

La cuenta *admin* nos abre todas las puertas de la red y hay que ser especialmente cuidadoso con ella. Así pues, debemos hacer algunas consideraciones importantes sobre el usuario *admin*:

- Tras la instalación es conveniente crear una cuenta con sus mismas equivalencias de seguridad y eliminar la cuenta *admin*. Mucho cuidado con esto, se trata de crear una cuenta con el mismo poder de *admin*, no de dar equivalencias de seguridad de *admin* a una cuenta y luego borrar *admin*. Si lo hiciéramos así se perderían todas las equivalencias de seguridad al borrar la cuenta de referencia y nos quedaríamos sin posibilidad de administrar la red. La razón de borrar la cuenta *admin* es no facilitarle la mitad del trabajo a un posible intruso en la red. Si ya sabe el nombre de la cuenta, tan sólo le queda averiguar la contraseña. Cuantas menos facilidades demos, mejor.

- Debemos ser muy cuidadosos con la contraseña del administrador. Casi tan malo como olvidarla es que se convierta en 'un secreto a voces'.

- Para evitar errores de graves consecuencias, resulta recomendable que el administrador de la red no entre habitualmente como tal. Lo mejor es crearse una cuenta normal de usuario y entrar al sistema como administrador sólo cuando sea necesario.

En grandes organizaciones, se crean tantas cuentas de subadminstradores como sean necesarias y posteriormente se apunta la contraseña del administrador y se guarda en lugar seguro. Sólo se volverá a utilizar en los casos estrictamente necesarios. Incluso se llega a fraccionar la contraseña y dar a conocer separadamente los fragmentos a varias personas, de modo que sólo una persona conozca la contraseña, y en su falta, sea necesario el acuerdo y la presencia de varios para entrar como administrador.

Seguridad en el acceso a la red

Respecto a las cuentas de usuario, el administrador o subadministrador puede imponer restricciones sobre la cuenta y sobre su propiedad; estas restricciones son:

- Restricciones de estación. Evita que el usuario pueda conectarse desde una estación que no sea la especificada.

- Restricciones de tiempo. El usuario sólo podrá conectarse a determinadas horas y días. Los días se expresan en días de la semana. Este tipo de restricción es útil para crear limitaciones del tipo 'de lunes a viernes de 8 a 3', como la que ilustra la figura 1.

- Restricciones de vencimiento. Establece una fecha y hora para la caducidad de una cuenta de usuario. Es muy útil para personas ajenas a la empresa (instaladores de *software*, por ejemplo) o trabajadores temporales. Si activamos esta restricción y se nos olvida eliminar la cuenta una vez han terminado su trabajo, lo más grave que puede pasar es que ocupen espacio en el disco de forma innecesaria.

- Restricciones de conexión. Limitan el número de conexiones concurrentes que un mismo usuario puede realizar desde diferentes estaciones.

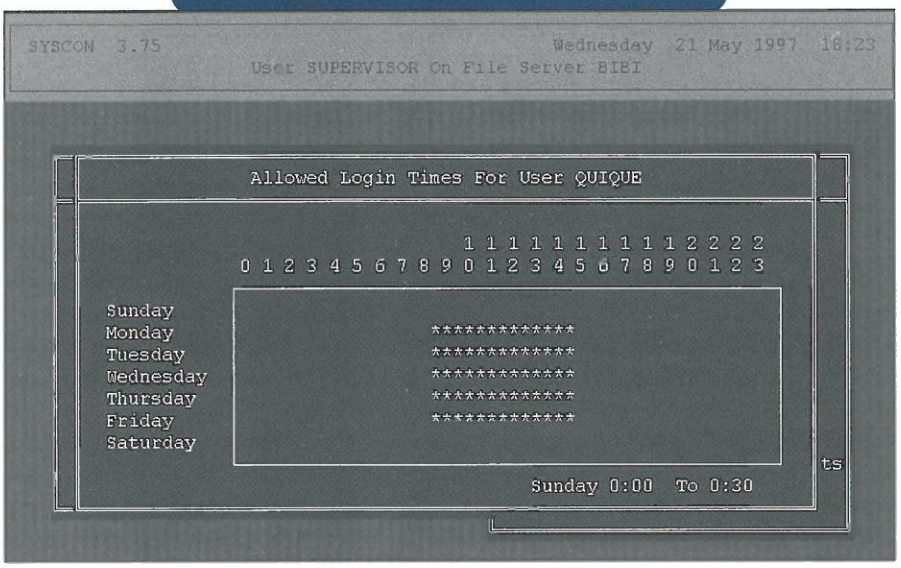
- Restricciones de espacio en disco. Se asigna al usuario un espacio máximo utilizable en disco.

- Restricciones de balance de cuenta. Para activar esta restricción es necesario haber instalado previamente el sistema de facturación de Novell. Consiste, básicamente, en otorgar al usuario un crédito que se va agotando en función del tiempo que esté conectado, de los recursos que utilice, o ambos. Una vez consumido el crédito el usuario no podrá conectarse de nuevo hasta que se le autoricen nuevos créditos.

- Restricciones de contraseña. El administrador puede especificar si la cuenta requiere o no contraseña, si caduca, o la longitud mínima de la contraseña. También se puede forzar a que se cambie la contraseña cada cierto tiempo e incluso obligar a que la nueva contraseña no haya sido utilizada recientemente.

- Detección de intrusos. Esta característica de seguridad de acceso a la red se activa en el contenedor de la cuenta (organización o unidad organizativa) y su efecto es bloquear una cuenta de usuario si se falla cierto número de veces la contraseña de acceso. Se puede especificar:
  • Activar o desactivar detección de intrusos.
  • Intentos Incorrectos de entrada. Es el número de veces que se podrá fallar la contraseña antes de bloquear la cuenta.

FIGURA 1


SYSCON 3.75
Wednesday 21 May 1997 16:23
User SUPERVISOR On File Server BIBI
Allowed Login Times For User QUIQUE
1 1 1 1 1 1 1 1 1 1 2 2 2 2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday 0:00 To 0:30

• Intervalo de restauración de intentos de intrusión. Es el tiempo que Netware 'recuerda' que se falló la contraseña. Transcurrido ese tiempo, el número de intentos restantes volverá a ser el original. Se pueden especificar días, horas y minutos.

• Intervalo de restauración de bloqueo de intrusos. Una vez se ha superado el número máximo de intentos, el sistema no permitirá la entrada a la red durante el tiempo que especifiquemos. Se pueden fijar días, horas y minutos.

La filosofía NDS

NDS (*Netware Directory Services,* servicios de directorio de Novell) es el servicio más importante proporcionado por las versiones 4 de Novell. Su cometido es proporcionar un punto único de acceso a la red, permitiendo el manejo de todos los recursos de la red de forma transparente.

El NDS es una base de datos llamada directorio, que se encuentra distribuida por todos los servidores y que contiene información sobre todos los objetos de la red.

Los objetos dentro del directorio se organizan de forma jerárquica y su estructura se representa en el llamado árbol de directorio, cuya misión es organizar de forma lógica los recursos de la red, lo que a menudo supone también una representación de la organización de nuestra empresa. De aquí se deriva la importancia de hacer un buen diseño del árbol de directorio, ya que con un buen diseño conseguiremos que la red sea fácil de administrar y de usar.

Cuando solicitamos acceso a un recurso, NDS localiza el recurso, comprueba si tenemos los permisos adecuados para usarlo comparando en una base de datos, y en caso afirmativo nos conecta a ese recurso. Los objetos tienen unas propiedades, que vendrían a ser los campos del registro de la base de datos. Estas propiedades varían de un objeto a otro, pero todos los objetos tienen una propiedad *nombre* que es lo que posibilita que el sistema pueda hacer referencia al objeto y acceder a sus propiedades. Los objetos del NDS pueden ser de tres clases *Root* (raíz), objetos contenedores y objetos hoja.

- **Root** se crea durante la instalación del primer servidor de la red y es el origen del directorio. Solo puede contener los objetos contenedores *País* y *Organización*. Esto quiere decir que no podremos colgar de la raíz unidades organizativas u objetos hoja. *Root* no se puede eliminar, mover ni crear. Pese a que evidentemente contiene otros objetos, no se le considera objeto contenedor debido a sus características especiales no compartidas por el resto de los objetos contenedores.

- **Objetos contenedores** son aquellos que contienen otros objetos, y pueden ser *País, Organización* y *Unidad Organizativa.*

  • El objeto *País* —no podía ser de otra forma— se utiliza para denominar países (no olvidemos que las versiones 4 de Novell pueden ser tanto LAN como WAN). Puede contener objetos hoja y la denominación de los países está limitada a dos letras.
  • El objeto *Organización* suele utilizarse para denominar las distintas compañías de un grupo de empresas. Es obligatorio designar al menos un objeto *Organización* en el árbol de directorios
  • El objeto *Unidad Organizativa* tiene la particularidad de ser el único que puede contener objetos de su mismo tipo, es decir, una unidad organizativa puede contener a su vez otras unidades organizativas. Se usa para designar grupos de trabajo, departamentos o proyectos.

- **Objetos hoja** son aquellos que no contienen otros objetos, tales como usuarios, servidores o impresoras. Representan generalmente los recursos de nuestra red.

Los objetos hoja son los siguientes:

• Alias. En terminología del OS/2 podría ser lo mismo que una *sombra* y en terminología Windows 95, un *acceso directo*. Es un puntero que apunta a cualquier otro objeto del árbol de directorios, de forma que ob-

jetos que no están en nuestra rama parecen estarlo.

• Asignación de Directorio. Representa un directorio en un volumen del servidor. Se emplea para especificar una vía de acceso a una aplicación.

• Bindery. Representa un objeto actualizado procedente de servidores Novell Netware cuya versión es anterior a la 4.

• Cola de Impresión. Representa el lugar donde se almacenan los trabajos antes de que el servidor de impresión pueda darles servicio.

• Computadora. Representa una estación de la red. No tiene uso efectivo, salvo el meramente informativo.

• Entidad Externa. Representa servicios ajenos al NDS.

• Grupo. Representa un grupo de usuarios y se suele utilizar para asignar derechos y condiciones que afecten a todos los integrantes del grupo.

• Grupo de Encaminamiento de Mensajes. Representa un grupo de servidores de correo que intercambian mensajes entre sí.

• Impresora. Representa un dispositivo físico de impresión.

• Lista de distribución. Es un objeto del Servicio de Gestión de mensajes (MHS). Se utiliza para enviar mensajes que serán recibidos por un grupo de usuarios.

• Perfil. Representa una secuencia de conexión que pueden ejecutar varios usuarios.

• Rol Organizativo. Representa un tipo de grupo con un cargo en la organización. Es utilizado para realizar la asociación de derechos especiales en función del cargo que representa.

Tabla 1. Reglas para crear el árbol NDS.

Objeto	Contenido en	Contiene
País	Root	Organización
Organización	Root y País	Unidad Organizativa, objetos hoja.
Unidad Organizativa	Organización y Unidad Organizativa	Unidad Organizativa, objetos hoja.

• Servidor AFP. Es un servidor de archivos Apple que funciona con el protocolo AppleTalk. No tiene efecto específico, salvo el meramente informativo.

• Servidor de comunicaciones. Representa un servidor de mensajes MHS.

• Servidor de impresión. Representa un servidor de impresión de la red.

• Servidor Netware. Representa un servidor cuyo sistema operativo es Netware y que proporciona servicios a la red.

• Usuario. Representa un usuario de la red. A través de este objeto se configura el acceso a la red y a sus recursos.

• Volumen. Representa un volumen de un servidor Netware.

El objeto Unidad Organizativa es el único que puede contener otros objetos de su misma clase

Existen unas sencillas reglas, detalladas en la tabla 1, para la creación de objetos dentro del árbol NDS. De acuerdo con estas reglas y las características de los objetos, el árbol más sencillo posible sería éste:

Root
  Organización
    Objetos Hoja

Y un árbol completo:
Root
  País
    Organización
      Unidad organizativa
        Objetos Hoja

Derechos de acceso

La seguridad de accesos en Novell Netware se basa en que los objetos se protegen más o menos solos, es decir, el sistema no 'sabe' permanentemente quién es quién y qué puede hacer sobre qué objetos (si fuera así se nos iría toda la memoria del servidor sólo en esto). La seguridad funciona más bien como un sistema de preguntas y respuestas. Cada vez que queremos hacer algo el sistema irá preguntando al objeto quiénes somos, a qué propiedades accedemos y qué autorizaciones tenemos sobre las propiedades, con esos datos en la mano se nos permitirá o no realizar determinadas operaciones. Para poder efectuar estas comprobaciones los objetos del árbol NDS tienen una lista de control de acceso o ACL (*Access Control List*) que contiene información sobre los derechos de objeto y los derechos de propiedad para cada miembro de la lista de acceso de ese objeto. Veamos esto más detenidamente. Los derechos que podemos otorgar sobre los objetos son:

PROGRAMAS MULTIMEDIA

SENSACIONAL OFERTA SÓLO PARA LECTORES DE SÓLO PROGRAMADORES

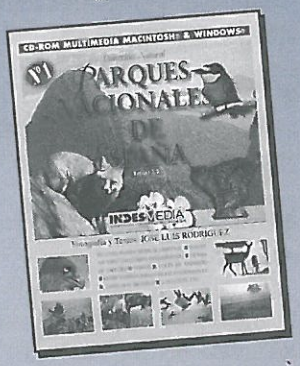
~~9.995~~ **AHORA 3.995 ptas. c/u**

ENCICLOPEDIA DE LA PINTURA

más completa galería de arte en CD-ROM. Permite acceder la información por pintores, estilos, temas, períodos, cionalidad de los pintores, ciudades y lugares donde se cuentran actualmente las obras. Incluye biografía del tor, características de la obra, medidas, etc.

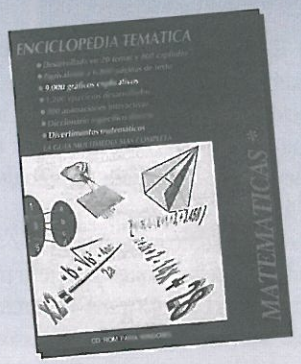
PLANTAS MEDICINALES

El Dioscórides con un nivel de interactividad absoluta, 600 Mb de información, 180.000 referencias alfabéticas, 2.000 imágenes, grabados y gráficos, descripción, uso, virtudes, etc.



PARQUES NACIONALES DE ESPAÑA

Parques Nacionales de España es una obra multimedia interactiva que facilitará a viajeros, amantes de la naturaleza, y usuarios en general, niños y mayores, el conocimiento de la fastuosa riqueza natural de España.

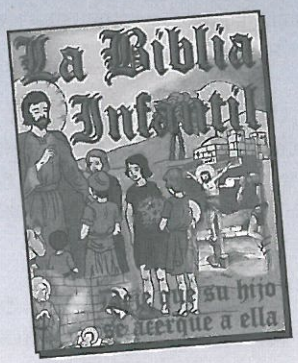


ENCICLOPEDIA MATEMÁTICA

meros, ecuaciones e inecuaciones, polinomios, geome-a, sistema métrico decimal, conjuntos, divertimentos ıtemáticos, personajes, etc., plasmados en 800 temas puestos en 2.800 pantallas,con más de 9.000 gráficos y 0 animaciones interactivas.

VINOS DE ESPAÑA

Más de 1.200 imágenes, 1.709 bodegas, 102 mapas autonómicos y denominación de origen, más de 1.000 plantillas de hipertexto, 23 secuencias de vídeo.



LA BIBLIA INFANTIL

Acerca a los más pequeños la religión católica, mediante ilustraciones interactivas, textos explicativos y locuciones hechas por profesionales. Una moderna herramienta de ayuda a la diversión, muy útil a la hora de hacer familia y un poco de apostolado.

VÁLIDO HASTA FIN DE EXISTENCIAS

Solicite los Programas Multimedia que desee, enviando este cupón por correo o fax (91) 661 43 86, o llamando al teléfono (91) 661 42 11*.
E-mail: suscrip@towercom.es Horario lunes a jueves de 9 a 14 y 15 a 18h. y viernes de 9 a 15h.

OLICITO LOS SIGUIENTES PROGRAMAS MULTIMEDIA:

-] Parques Nacionales de España
-] Enciclopedia Matemática
- ☐ Enciclopedia de la Pintura
- ☐ Plantas Medicinales
- ☐ La Bibilia Infantil
- ☐ Enciclopedia de los Vinos de España

ombre y apellidos ..... F. Nacimiento .....
omicilio ..... C.P .....
el. ..... Localidad .....
rovincia ..... Profesión .....

ORMA DE PAGO:

] Con cargo a mi tarjeta VISA nº..... .... Firma:
Fecha de caducidad de mi tarjeta..... / .....
] Contra-reembolso del importe más gastos de envío y embalaje 450 ptas.
] Cheque a nombre de CD Proyectos Especiales que adjunto.
] Giro Postal (adjunto fotocopia del resguardo).

TOWER COMMUNICATIONS S.R.L.
APTDO. CORREOS 54.283
28080 MADRID

- *Browse* (buscar), que permite que los usuarios vean el objeto en el árbol de directorios, pero nada más.

- *Create* (crear), que permite al usuario nuevos objetos dentro de un objeto contenedor.

- *Delete* (borrar), permite a los usuarios borrar objetos del árbol de directorios.

- *Rename* (renombrar), permite renombrar objetos.

- *Supervisor*, equivalente a todos los anteriores.

Habitualmente los usuarios normales sólo necesitarán el derecho *Browse* para observar los objetos en el árbol de directorios. El resto de los derechos se lo otorgaremos a personas que tengan responsabilidades de administración en la red.

Como hemos dicho antes, existen derechos de objeto y derechos sobre las propiedades de los objetos. Los derechos sobre las propiedades de los objetos son éstos:

- *Compare* (comparar) devuelve verdadero o falso. Esta propiedad se supone implícitamente en el derecho *read*.

- *Read* (leer) permite ver el valor de una propiedad.

- *Write* (escribir) permite cambiar el valor de una propiedad.

- *Add/Delete Self* (añadirse o eliminarse) permite añadirse o eliminarse de una propiedad. Esta propiedad sólo es válida para propiedades que contienen nombres de objeto como valores, como por ejemplo pertenencia a grupos o listas de correo.

- *Supervisor* es el equivalente a todos los derechos comentados anteriormente.

Esto parece C

En principio el tema de los derechos de objeto y las propiedades de los objetos puede parecer muy confuso (concretamente un servidor jamás ha visto a nadie que lo haya entendido a la primera). Por ello vamos a aclararlo definitivamente.

Haciendo un símil para los que sepan algo de lenguajes de programación orientados a objeto como C++ o Java, los objetos del árbol NDS tienen una serie de propiedades. Y cuando otorgamos derechos sobre las propiedades lo que hacemos es dar al usuario el acceso a métodos sobre esas propiedades, es decir, le damos acceso a métodos para leer el valor de esa propiedad, para cambiar el valor de esa propiedad, etc.

Veamos un ejemplo con la aplicación efectiva de todo esto a la seguridad dentro de la red.

Tomemos un objeto usuario. Ese objeto tendría unas propiedades determinadas, como nombre de la cuenta, nombre completo, número de teléfono, restricciones de la conexión, restricciones de tiempo, entre otras posibles. Al propio usuario le podríamos otorgar derechos de escritura sobre diferentes propiedades como podrán ser el nombre, la dirección y el teléfono, mientras que un administrador tendría derechos de escritura sobre las propiedades de restricción de conexión, restricción de tiempo, número de teléfono, etc.

Seguridad del sistema de archivos

El árbol de directorios NDS y el sistema de archivos de Netware son completamente distintos, sin embargo, comparten las mismas técnicas de administración y tienen dos puntos de unión:

1. Un usuario con derecho de supervisor sobre un objeto contenedor tiene derechos de supervisor también en el sistema de archivos de los servidores que estén dentro de ese contenedor.

2. Todos los usuarios tienen derechos de lectura y búsqueda en el directorio SYS:PUBLIC de cualquier servidor.

Veamos ahora cómo nos afectan estos puntos de unión con respecto a los derechos sobre directorios y archivos (la lista de derechos sobre directorios y archivos aparece en la tabla 2):

- Un usuario con derecho de supervisor sobre un objeto contenedor que contenga objetos servidor, tiene derecho de supervisor sobre todos los volúmenes de todos los servidores contenidos en el objeto.

- El objeto public tiene derecho a examinar sobre la raíz del árbol de directorios. Por defecto todos los usuarios tienen este derecho y por tanto pueden ver los objetos contenidos en el árbol de directorios.

- Los derechos que un objeto pueda tener sobre el sistema de archivos, sólo se modifican por asignación directa del supervisor o a través del filtro de derechos heredados.

El filtro de derechos heredados

El filtro de derechos heredados (IRF, *Inherited Rights Filter*) actúa como un 'colador' de derechos y lo hace tanto a nivel de NDS como de sistema de archivos. Recordemos que en Novell Netware obtenemos los derechos por asignación directa del administrador y por la herencia que se deriva de estas asignaciones (también por equivalencias

Tabla 2. Derechos sobre directorios y archivos.

Derechos	Efectos sobre directorio	Efectos sobre archivo
R Read (lectura)	Permite abrir, leer y ejecutar archivos, si la propia máscara . del archivo lo permite	Permite abrir y leer archivos.
F File Scan (Buscar archivo)	Permite ver los archivos que hay dentro del directorio, si . la máscara del archivo lo permite	Permite ver el archivo al listar el contenido de su directorio. Junto con Read permite ejecutar archivos.
M Modify (modificar)	Permite renombrar el directorio y modificar sus atributos, pero no a modificar el directorio en sí. Hay que recordar que este derecho se hereda para los archivos y subdirectorios contenidos.	Permite renombrar archivos y modificar sus atributos, pero no su contenido. Tampoco permite leerlo.
W Write (escritura)	Permite abrir y modificar el contenido los archivos del directorio.	Permite abrir y modificar el contenido del archivo.
C Create (crear)	Permite crear archivos y subdirectorios. Pese a que parezca que este derecho implica escribir leer y modificar, no es así. Si sólo otorgamos este derecho el usuario no podría leer ni modificar un archivo creado por el mismo tras haberlo cerrado.	Permite recuperar archivos borrados.
E Erase (borrar)	Permite borrar directorios y todo su contenido, salvo que la máscara de algún archivo lo impida.	Permite borrar el archivo.
A Access Control (control de accesos)	Permite modificar la lista de control de acceso y la máscara de derechos heredados del directorio y de todo su contenido. Permite asignar y revocar todos los derechos, excepto el de supervisor.	Idéntico, pero sólo para el archivo en particular.
S Supervisor	Asigna todos los derechos anteriores sobre el directorio y todo su contenido. Prevalece sobre la máscara de derechos heredados, de modo que no puede ser filtrado. De este modo el derecho de supervisor sólo puede ser eliminado del directorio donde se asignó originalmente. Es decir, no se puede otorgar el derecho de supervisor en un directorio y eliminarlo de otro contenido en ese directorio, ya que se ignoran los filtros situados por debajo.	Asigna todos los derechos anteriores sobre el archivo, y no puede ser bloqueado por ningún filtro de derechos

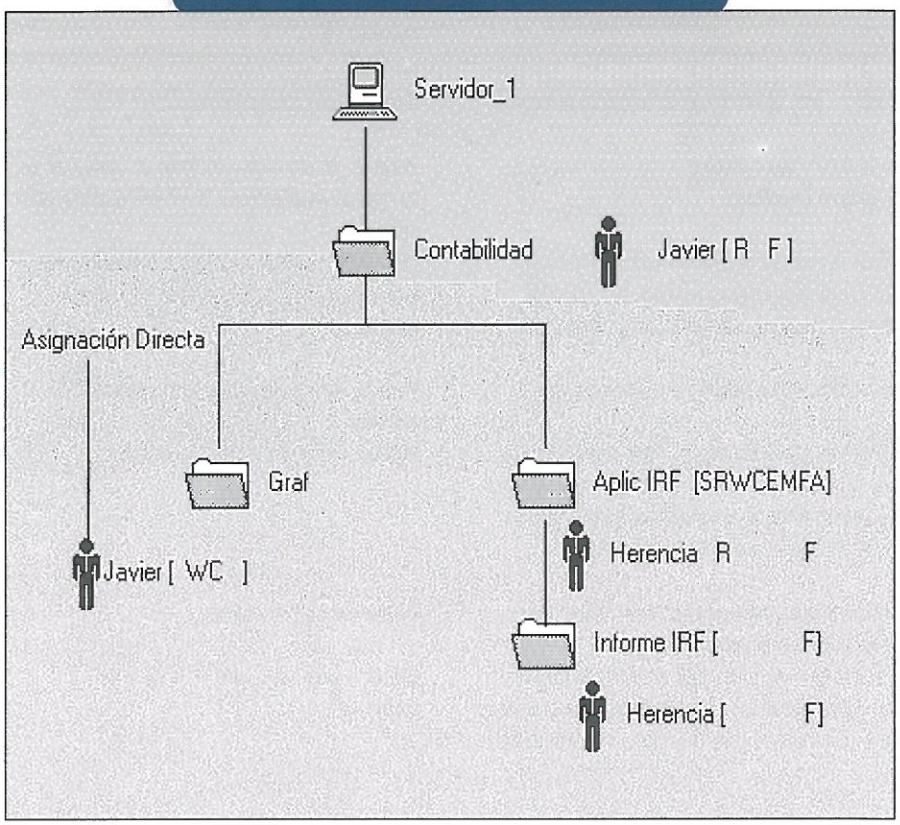
de seguridad, pero eso lo veremos más adelante). El filtro de derechos heredados sirve para controlar los derechos que los usuarios pueden heredar de los objetos contenedores padre en el árbol NDS o de los directorios padre del sistema de archivos. El IRF se utiliza para bloquear derechos que se heredan desde un objeto o directorio padre.

Veamos un ejemplo: en la figura 2 el usuario Javier tiene derechos de lectura y exploración sobre el directorio contabilidad. El IRF del directorio aplic deja pasar todos los derechos, así pues Javier tendrá también los derechos de lectura y exploración sobre el directorio aplic. En cambio el IRF del directorio informe sólo deja pasar el derecho de exploración, con lo que Javier perderá el derecho de lectura sobre este directorio. En el directorio graf, Javier obtiene los derechos de escritura y creación por asignación directa del administrador, con lo que la herencia deja de tener efecto para este directorio, y Javier sólo tendrá los derechos que le ha otorgado explícitamente el administrador.

Con respecto al IRF podemos dar las siguientes reglas:

- Cada objeto, directorio y archivo puede tener su propio IRF.
- Los objetos dentro de contenedores obtienen derechos a partir del contenedor mediante equivalencia de seguridad; estos derechos no pueden bloquearse a través de IRF.
- El IRF no tiene efecto si un usuario tiene derechos explícitos sobre un objeto, o sea, si el administrador le ha hecho miembro de la lista de acceso de ese objeto.
- En el sistema de archivos, el derecho de supervisor se hereda para todos los subdirectorios y archivos, y no puede ser filtrado por el IRF

FIGURA 2



(el derecho de supervisor sólo puede eliminarse del directorio donde se concedió). En cambio, en el árbol NDS el IRF sí puede bloquear el derecho de supervisor concedido en objetos superiores.

Herencia y derechos efectivos

Supongamos que hemos concedido a un usuario derechos de exploración, lectura, creación, borrado y renombrar sobre un objeto contenedor para que pueda administrar las cuentas de usuario. Estos derechos se extenderán desde el objeto contenedor hacia abajo sobre todos los objetos dentro de él. Ya que queremos que el usuario administre las cuentas de usuario y no los otros objetos del contenedor (por ejemplo servidores) podemos bloquear estos derechos en los objetos donde interese con el IRF.

Entender bien este concepto de flujo de derechos es muy importante en el sistema de seguridad de Novell. Un punto que a menudo se olvida y que da lugar a fallos en la seguridad, son los derechos por omisión, que se otorgan automáticamente al supervisor y a los usuarios. Los derechos por omisión son los siguientes:

- Existe una entidad llamada *public* que tiene derecho a examinar en la raíz del árbol NDS y todos los usuarios que se crean tienen por defecto equivalencias de seguridad a *public*. Por tanto, cualquier usuario tiene derecho a examinar la red incluso antes de haberse conectado (por eso funcionan los mandatos *cx* antes de hacer el *login*). Si se desea evitar esto hay que revocar este derecho de la entidad *public* y otorgarlo al contenedor de organización de más alto nivel. Así los usuarios podrán examinar el árbol sólo después de haberse conectado a la red. La entidad *public* también tiene derechos de lectura sobre la propiedad *Network Adress* de los servidores.

- El usuario *admin* tiene derecho de supervisor sobre la raíz del NDS y, por tanto, controla el árbol y el sistema de archivos. En el caso del sistema de archivos el derecho de supervisor es el único derecho de NDS que se traslada al sistema .

- Los contenedores que contengan servidores tendrán derechos de exploración y lectura sobre el directorio SYS:PUBLIC de todos los servidores contenidos. Las cuentas de usuario consiguen esos derechos por equivalencia de seguridad.

- La raíz del árbol NDS tiene derechos de lectura sobre las propiedades *Network Adress, Group Membership* y *Default Server* de las cuentas de usuario

- Los usuarios tienen derechos de lectura y escritura sobre sus propias secuencias de conexión y sobre las propiedades de configuración de los trabajos de impresión que estén entre las propiedades de sus cuentas.

Existe, además, otra forma de conseguir derechos mediante las equivalencias de seguridad. La equivalencia de seguridad es posible aplicarla tanto a los usuarios como a los objetos y consiste básicamente en proporcionar al usuario u objeto los mismos derechos que tiene el objeto o usuario al que le hacemos equivalente.

Las equivalencias de seguridad constituyen un método rápido de otorgar derechos a un usuario y objetos. Supongamos que se ha contratado a un nuevo administrativo en la empresa y debemos crearle una cuenta de usuario. Bastaría con rellenar el campo de *Login Name* y darle las equivalencias de segu-

Tabla 3. Atributos de archivos y directorios

Atributo	Efecto
A (Respaldar)	(Sólo archivos) Si está activo, indica que el archivo ha sido modificado desde la última copia de seguridad.
CI (Impedir copia)	(Sólo archivos) Evita la copia del archivo por parte de los usuarios de Macintosh.
DI (Impedir borrado)	Evita el borrado de un archivo o directorio.
DC (No comprimir)	Evita que el archivo sea comprimido. Si se aplica a un directorio evitará que los archivos del directorio sean comprimidos.
DS (No subasignar)	(Sólo archivos) Evita que el archivo se subasigne, aunque esté activada la característica de subasignación en el volumen. Suele aplicarse para archivos de base de datos o archivos que aumentan de tamaño con cierta frecuencia.
DM (No migrar)	Evita que un archivo o el contenido de un directorio sea migrado a un dispositivo de almacenamiento secundario, como por ejemplo un disco óptico.
X (Sólo ejecución)	(Archivos) Impide que se copie un archivo. Este atributo sólo puede ser activado por el supervisor, y una vez activado, no puede suprimirse. Se debe usar sólo si existe una segunda copia del archivo, ya que las utilidades de copia de seguridad no hacen copia de estos archivos. Algunos archivos de programa no funcionan correctamente si se les aplica este atributo.
H (oculto)	Oculta directorios y archivos al hacer un dir. También evita que el archivo o directorio sea copiado o eliminado.
IM (Compresión inmediata)	El archivo o el contenido del directorio serán comprimidos tan pronto como sea posible.
P (Borrado definitivo)	Los archivos o los directorios y sus contenidos son eliminados definitivamente cuando se borran. Los archivos borrados que tengan este atributo activo no podrán recuperarse con las utilidades de recuperación de archivos de Netware.
RO (Sólo lectura)	(Archivos) Impide que un archivo se modifique, renombre o suprima. Los atributos RI y DI se aplican automáticamente al activar RO.
RI (Impedir renombrado)	Evita que un archivo o directorio sea renombrado.
S (Compartido)	(Sólo archivos) Permite el acceso simultáneo de varios usuarios a un archivo. Se aplica generalmente para archivos de base de datos con bloqueo de registros.
SY (Sistema)	Evita que los usuarios vean el archivo o directorio al hacer un dir.
T (Transaccional)	(Sólo archivos) El sistema de seguimiento de transacciones protegerá al archivo. Se usa en archivos de base de datos.
CC (No comprimir)	Este atributo le indica al sistema que el archivo no puede ser comprimido para ahorrar espacio en disco.
C (Comprimido)	Lo asigna automáticamente el sistema para indicar que el archivo ha sido comprimido.
M (migrado)	Lo asigna automáticamente el sistema para indicar que el archivo ha sido migrado.

ridad de cualquier otro administrativo que ya trabaje con nosotros. Sin embargo esta forma de otorgar derechos no resulta recomendable ya que si se elimina la cuenta de origen todos los objetos y usuarios perderán los derechos equivalentes. Es muy útil aprovechar las equivalencias de seguridad del objeto contenedor, ya que todos los objetos dentro de él adquieren equivalencias de seguridad de su contenedor. Así cualquier derecho que otorguemos al objeto contenedor será adquirido por los objetos contenidos en él. Igual ocurre cuando hacemos que un usuario se convierta en miembro de un grupo, ya que el nombre del grupo se añade a la propiedad *Security Equal To* de la cuenta de usuario. Esto simplifica la administración, ya que si alguien deja de pertenecer a un grupo o le cambiamos de contenedor pierde los derechos derivados de ello. De igual forma adquiere los nuevos derechos inherentes a su contenedor o grupo.

Lo explicado nos lleva a lo que importa, los derechos efectivos, lo que puede o no hacer el usuario en la red. Estos derechos dependerán de las asignaciones directas o explícitas hechas por el administrador, añadiéndonos a la lista de acceso de los objetos; de las equivalencias de seguridad con otros usuarios u objetos; de los derechos que se heredan de niveles superiores, de los de los grupos a los que pertenezcamos y de los filtros de derechos heredados, que pueden restringir nuestros derechos heredados. También hay que advertir que los derechos de un usuario no siempre aparecen de forma clara en los datos de la cuenta de usuario, ya que allí sólo veremos los derechos explícitamente otorgados por el administrador, sus equivalencias de seguridad y los grupos a los que pertenece. Esto significa que deberemos investigar y ver cuáles son los derechos que obtiene de la pertenencia a grupos, los que obtiene por herencia y los que obtiene por equivalencia de seguridad con su contenedor.

VISUAL BASIC AVANZADO

Programación OLE

Juan Manuel Menéndez

Finalizando esta serie sobre programación OLE con Visual Basic vamos a utilizar sus herramientas para la creación de servidores OLE tanto internos como externos. También veremos algunos enfoques generales para utilizar esta interfaz de programación en el diseño de aplicaciones.

En los artículos anteriores de esta serie de Sólo Programadores se ha estado programando desde la parte del cliente, accediendo a servidores que nos mostraban parte de su código por medio de una interfaz. En este artículo vamos a cambiar y diseñar nuestro propio servidor. Una de las novedades que traía Visual Basic 4 en sus ediciones Profesional y Empresarial era la posibilidad de crear servidores que pueden ser utilizados desde otros clientes escritos en otros lenguajes, como C++.

Concepto de objeto

En el área de la programación, un objeto es un conjunto de instrucciones (un programa) diseñado para realizar una tarea que presenta sus componentes por medio de una interfaz estándar de programación.

Los objetos de un mismo tipo pertenecen a una clase o, dicho de otra forma, un objeto es una particularización del modelo general (clase). En un lenguaje más técnico, los objetos son instanciaciones de las clases.

Para poner un ejemplo de la relación clase/objeto aplicado en programación clásica, la clase corresponde a la definición de una estructura de datos en memoria y el objeto es el área de memoria que contiene los datos representados definidos en la estructura. Al igual que pueden existir varias áreas de memoria con la misma estructura y cada una de ellas con diferentes valores, pueden existir en un programa varias instanciaciones de una clase (objetos) y cada una de ellas con distintos valores.

Las estructuras de datos tienen componentes que son las distintas áreas de datos que la forman y cada área de datos tiene sus propiedades, como pueden ser: tipo de datos, longitud, rango, etc. Los componentes de una clase son los métodos y las propiedades. Las propiedades de una clase realizan un papel similar al de los campos en las estructuras de memoria, mientras que los métodos son las tareas que puede realizar una clase y que determinan las habilidades de la misma. Podemos encontrar un objeto de clase válvula con los siguientes componentes:

- Material, diametro.
- AbrirValvula, CerrarValvula, CalcularConsumo.

Material y diámetro son propiedades de la clase, mientras que AbrirValvula, CerrarValvula y CalcularConsumo son procedimientos. Podemos disponer de varios objetos de la clase Válvula cada uno con distintas propiedades, como pueden ser válvulas de distinto diámetro o material, pero todas ellas realizan las mismas

funciones (métodos). Los componentes de una clase a su vez se dividen en:

- Públicos: Pueden ser accedidos desde fuera de la clase.
- Privados: No pueden ser accedidos desde fuera de la clase.

Según lo anterior, la interfaz que presenta un servidor de objeto a sus clientes son los componentes públicos.

Todavía no se puede considerar a Visual Basic como un auténtico lenguaje orientado a objetos, ya que carece de todas las funcionalidades que debe tener un lenguaje de este tipo, entre las más importantes cabría reseñar:

- Herencia
- Sobrecarga de operadores.
- Atributos de los componentes (además de Público y Privado)

La versión 5 de VB incrementa las características de POO, dentro de lo que parece ser la política de Microsoft de llegar a convertir Visual Basic en un auténtico lenguaje orientado a objetos.

Creando un servidor

Un servidor OLE en Visual Basic se comienza con la creación de un proyecto nuevo y seleccionado en el menú Herramientas/Opciones en la pestaña Proyecto el Modo de Arranque (Start Mode) que vamos a crear un servidor OLE.

Hasta la aparici;on de la versión 4 de Visual Basic existían dos clases de ficheros con código:

- Módulo formulario (.frm)
- Módulo de código (.bas)

Visual Basic 4 ha añadido a estas clases de ficheros los módulos de clase (.cls). Cuando se crea un servidor de OLE lo primero que debemos crear en nuestro proyecto es un módulo de este tipo. Una vez hemos abierto un nuevo proyecto de Visual Basic, seleccionaremos en el menú la opción Insertar y dentro de este menú seleccionar Módulo de clase. Nos aparecerá una nueva ventana de código de Visual Basic, este módulo tiene sólo tres propiedades : Nombre, Instancing y Public. Las dos últimas marcan el comportamiento de la clase.

La propiedad *Public* indica si la clase es visible fuera del proyecto. Poniendo el valor True a esta propiedad la clase es visible desde fuera del proyecto. Los lectores que estén siguiendo el artículo y dispongan de la versión Standard de Visual Basic sólo podrán crear objetos visibles desde dentro del proyecto. El acceso a objetos externos al programa se realiza por medio de Automatización OLE y sólo las versiones Profesional y Empresarial de Visual Basic disponen de esta funcionalidad. Si se desea realizar acceso a objetos internos desde la versión estándar únicamente, es necesario añadir los módulos de clase necesarios al proyect y trabajar con ellos como si fueran externos.

La propiedad *Instancing* indica si el objeto puede ser creado desde fuera del proyecto, esta propiedad tiene tres valores:

0: No se puede crear desde fuera del proyecto.

1: Se puede crear desde fuera del proyecto, pero únicamente habrá una copia en ejecución en todo el sistema. Si no existiera se arrancaría pero si existiera entonces se utilizaría esa copia.

2: Cada vez que se cree un objeto de esta clase se arrancará una copia del mismo.

Estas opciones tampoco están disponibles en la versión estándar.

La otra propiedad *Name* (Nombre) nos permite personalizar nuestra clase. Esta propiedad junto con el nombre del proyecto es lo que identifica a nuestro servidor OLE en el Registry. En nuestro ejemplo el proyecto tiene el nombre servidorPruebas y la clase sMensajes; disponiendo de estos valores el servidor se identificará en el Registry mediante el nombre compuesto por ambos servidorPruebas.sMensajes, tal como podemos observar en la figura 1 .

El acceso a objetos externos al programa se realiza por medio de Automatización OLE y sólo las versiones Profesional y Empresarial de Visual Basic disponen de esta funcionalidad

Figura 1

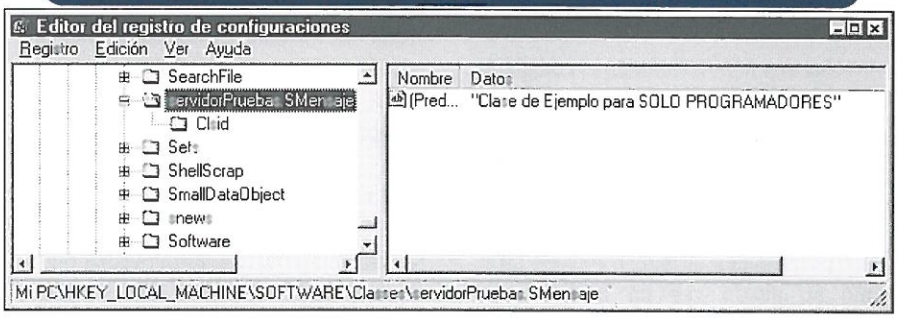


Figura 2

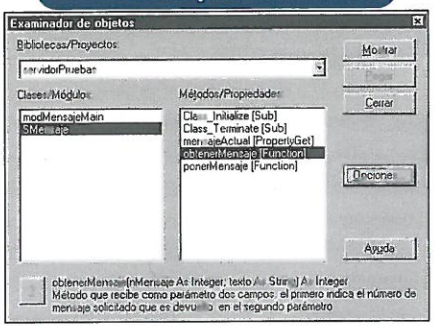
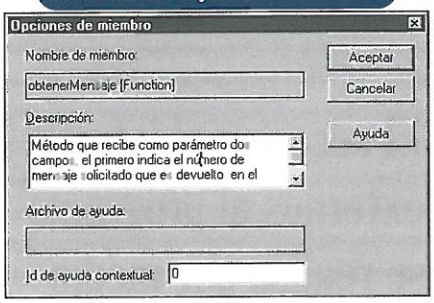


Figura 3



Con estos pasos ya hemos creado nuestra clase, que deberemos rellenar con métodos y propiedades. El servidor que se va a diseñar recibirá un texto del cliente y lo pondrá en una cola de mensajes, de la que se podrá extraer un mensaje determinado indicando el número de mensaje, o bien el último mensaje introducido. Para realizar estas tareas se han definido 3 métodos y dos propiedades. Los métodos son:

- ponerMensaje
- obtenerMensaje
- obtenerUltimoMensaje
- Y las propiedades son:
- Número de mensajes en el servidor
- Máximo número de mensajes

Una vez hemos definido el código de nuestros métodos, podemos ver cómo queda nuestro servidor utilizando el Examinador de objetos (Menú Ver/Examinador de Objetos). Este cuadro de diálogo nos indica cuáles son los métodos y propiedades de nuestra clase, tal y como se puede ver en la figura 2.

La destrucción de un objeto se realiza cuando su cuenta de uso de pone a cero

Podemos conseguir un toque más profesional en nuestra aplicación si suministramos información a cada elemento de la clase; para ello seleccionamos el elemento deseado en la caja de lista Metodos/Propiedades y pulsamos el botón opciones. Aparecerá una ventana como la de la figura 3, donde podremos escribir un pequeño texto informativo.

Los módulos de clase disponen, además, de dos procedimientos de evento asociados a ellas, que son los procedimientos *Initialize* y *Terminate*. El procedimiento *Initialize* se ejecuta cuando el servidor se arranca mediante la función *CreateObject*; en este procedimiento podremos colocar todas las sentencias necesarias para la inicialización de nuestro módulo de clase, obtención de perfiles de ejecución, etc. En el procedimiento *Terminate* que se ejecuta cuando el objeto se destruye, se colocarán las sentencias necesarias para el correcto cierre del servidor.

La destrucción de un objeto se realiza cuando su cuenta de uso de pone a cero. Veamos esto un poco más en detalle. Se puede hacer referencia a los objetos varias veces desde el mismo programa o desde otro programa. Cada vez que se hace referencia al objeto su cuenta de uso se incrementa en una unidad y cada vez que la variable elimina esa referencia, bien explícitamente, poniendo su valor a *Nothing*, o bien implícitamente, al desaparecer el ámbito de la misma, ese contador se decrementa en uno. El objeto se destruye cuando todas las referencias a él desaparecen, o lo que es lo mismo, cuando su cuenta de uso toma el valor cero.

El servidor que hemos creado dispone además de un formulario que nos servirá para comprobar de forma visual que se ejecuta; cambiando en el cuadro inicial el formulario de inicio a la rutina *Sub Main*, anulamos la visualización del formulario, obteniendo un servidor que trabaja de forma invisible. Hay que tener en cuenta que los formularios visibles a través de los módulos de clase no se descargan automáticamente al destruirse el objeto, es necesario hacerlo explícitamente en el procedimiento *Terminate*; de lo contrario quedarán en el sistema formularios cargados, con el consiguiente consumo de recursos.

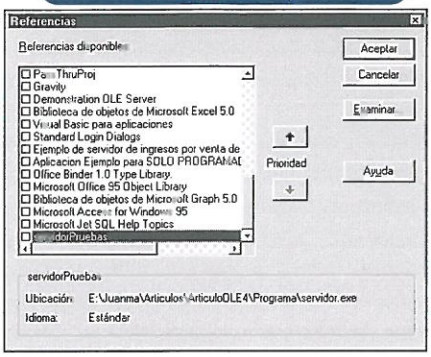
Una vez se ha codificado nuestro programa, hay que compilarlo y crear el ejecutable. Con Visual Basic disponemos de dos formas de compilar un servidor OLE como servidor *in-process* (en forma de DLL) o como *cross-process* (en forma de EXE). Estas dos formas están contempladas en el menú Archivo con las opciones Crear Archivo DLL de OLE y Crear Archivo EXE respectivamente; seleccionamos la opción para crear un archivo EXE (servidor *cross_process*). Una vez creado nuestro ejecutable podemos ver los resultados en el *Registry*, tal y como se ha explicado anteriormente.

Creando el cliente

Una vez tengamos nuestro servidor, procederemos a desarrollar el programa cliente. Para ello, crearemos un nuevo proyecto al que añadiremos las definiciones de nuestro servidor. En el menú Herramientas seleccionamos Referencias y aparecerá el cuadro de dialogo Referencias Disponibles (Figura 4); seleccionamos nuestro servidor marcando la *check box* servidorPruebas. A partir de este momento, toda la información disponible del servidor se encuentra en el examinador de objetos, información que se ha creado en el diseño de objeto (Figura 2). La codificación se vio en el articulo anterior por lo que no entraremos a describirla. Primero se crea el objeto con la función *CreateObject* o con el verbo *New*:

```
Set objetoMensajes = New SMensaje
Set objetoMensajes =
        CreateObject("servidorPruebas.sMensaje")
```

Figura 4



Este programa cliente se compone de un formulario con 3 botones; cada botón llama a uno de los métodos del objeto. En el artículo anterior se explicaba con detalle cómo realizar este tipo de programas, por lo que remitimos al lector a dicho artículo para seguir su código.

Depuración del código

Al diseñar una aplicación que proporciona un objeto OLE, nos encontramos ante una nueva situación: hay que depurar un programa que no interacciona con el usuario a través de un formulario como viene siendo habitual en los programas desarrollados en Visual Basic. El programa que tenemos que depurar recibe mensajes de otros programas a través de llamadas del sistema operativo. El entorno de programación de Visual Basic 4 tiene contemplada esta situación y el primer requisito es tener abiertos dos proyectos simultáneamente, cada uno en una ejecución distinta de Visual Basic, por lo que necesitaremos ejecutar dos veces VB. El primer paso para probar nuestro código es abrir el proyecto del servidor; marcaremos nuestros puntos de *break* y pulsaremos el botón de ejecución. El entorno de programación crea las entradas necesarias en el sistema para que nuestro objeto sea reconocido. A continuación ejecutamos otra instancia de VB con el proyecto cliente. Antes de ejecutar el código cliente deberemos asegurarnos en los Opciones del Proyecto (Pestaña Avanzada) que la interceptación de errores está Parada en módulo de clases. Con esta opción logramos hacer que el servidor OLE se interrumpa en sus módulos de clase, en lugar de devolver siempre el error al programa cliente. Ejecutamos el proyecto cliente y podemos empezar a depurar nuestro código igual que si estuviéramos depurando un programa simple, salvo que nos estamos moviendo entre dos instaciaciones de VB; obviamente si nuestra aplicación tuviera más servidores trataríamos con más proyectos abiertos (a base del *Alt Tab*), cosa bastante engorrosa con el entorno de programación de la versión 4. Este inconveniente se soluciona con la 5, que nos permite tener varios proyectos abiertos en la misma ejecución del VB y el entorno viene integrado en un formulario MDI y las demás ventanas como hijas.

Diseño de aplicaciones con OLE en Visual Basic

El uso de OLE desde VB admite dos puntos de vista: el de la arquitectura de las aplicaciones y el de la realización de proyectos de desarrollo de aplicaciones.

En la realización de aplicaciones, podemos añadir controles contenedores OLE que activen aplicaciones auxiliares, evitando que el usuario tenga que volver al escritorio o dirigir al mismo en el uso de un determinado componente (figura 5).

Figura 5.

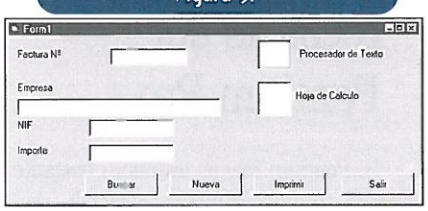
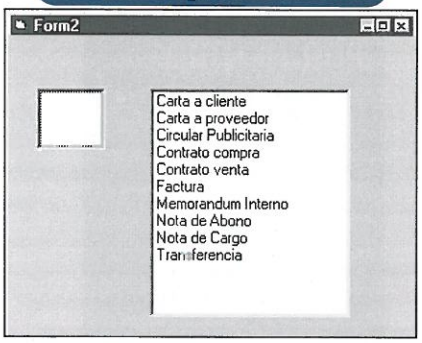


Figura 6



Se pueden crear sencillas aplicaciones gestoras de documentos; por ejemplo, desde una lista de documentos (figura 6) activar la aplicación contenedora OLE de la siguiente manera:

```
Private Sub List1_DblClick()
OLE1.DataSource = Documento.seleccionado.
End Sub
```

Mediante OLE Automatización podemos utilizar los elementos de la interfaz para crear aplicaciones de gestión utilizando las herramientas del mercado. Igualmente podemos crear servidores OLE para la implementación de servidores de negocio según el modelo cliente/servidor en 3 capas. No sólo accesibles desde otros ejecutables VB, sino también desde formularios Access, hojas de cálculo, etc. Además, se pueden crear sistemas completos de gestión utilizando componentes ya existentes en el mercado o de creación propia.

En la gestión de proyectos conseguimos:

- Componentes OLE que suministren servicios a otras aplicaciones.
- Dividir el código en diversos proyectos entre varios equipos y realizar la integración de los mismos mediante OLE bien como clases internas o externas.
- Creación de nuevos tipos de datos.
- Mejora del rendimiento utilizando servidores *cross-proccess* (en determinadas aplicaciones).

EL PC
POR DENTRO

Análisis del Pentium Pro

Jorge Figueroa

El Pentium Pro, pese a que parecía haber sido olvidado debido a la incorporación en el mercado hace algunos meses de los Pentium MMX, está destinado, gracias a la reciente aparición del Pentium II (un Pentium Pro MMX), a convertirse en el substituto de todos los chips actuales, ya que posee muchas características mejoradas en todos los aspectos.

Intel Corporation acaba de presentar la nueva generación de microprocesadores conocida ya como Pentium II que es, a grandes rasgos, un Pentium Pro al que se le han añadido también las 57 instrucciones MMX que podemos encontrar desde hace algún tiempo incorporadas en el Pentium MMX. Esto, junto a las modernas características de arquitectura que posee este micro y la velocidad a la que correrá (más de 200 Mhz), hace que esté destinado a convertirse en el micro que se instalará en todos los ordenadores de un futuro más próximo de lo que parece.

Mientras se espera que se introduzca definitivamente en el mercado este llamado Pentium II (conocido internamente hasta ahora como Klamath) y del cual Intel mostró en la *International Solid State Circuits Conference* de San Francisco un prototipo que podía funcionar a 433 Mhz, diseñado con tecnología de 0.35 micras y formado por 7.5 millones de transistores, vamos ahora a detallar las características, instrucciones y todo lo que destaca del actual Pentium Pro.

Características técnicas del Pentium Pro

El Pentium Pro es un micro bastante avanzado tecnológicamente hablando respecto a los anteriores microprocesadores de la misma casa. A continuación detallamos los datos que lo caracterizan:

• Diseño:

Está diseñado en una caja de cerámica del tipo PGA y posee 387 Pins. Ocupa 306 milímetros cuadrados y la caché Level 2, que en el Pentium Pro está integrada en el encapsulado del Chip, ocupa 202 milímetros cuadrados aparentando que en el encapsulado hay dos microprocesadores.

La caché level 2 del Pentium Pro está integrada en el chip

• Tecnología de fabricación:

Este microprocesador está formado por un total de 5.5 millones de transistores con una escala de integración de 0.6 micras en BiCMOS de cuatro capas. Requiere para funcionar 2.9 V.

• Microarquitectura:

El Pro está formado por un total de cinco unidades de procesamiento paralelo, siendo una para mediar con la memoria, dos para enteros y dos para reales (copro-

cesador) que forman parte de las doce etapas de procesado en que está diseñado el micro.

El Pentium Pro está formado por 5.5 millones de transistores

Por otra parte, el Pentium Pro está basado en la llamada tecnología *Dynamic execution*, concepto que ampliaremos más adelante y que consiste principalmente en que el puntero de instrucción no es una sóla dirección de memoria, sino que hay una ventana virtual donde van ejecutándose las instrucciones, consiguiendo de esta forma que no se interrumpa la ejecución en paralelo en caso de encontrar instrucciones interdependientes, ya que el procesador las salta para buscar la siguiente que pueda ejecutar dejando pendientes las que requieren como parámetros de entrada resultados de instrucciones que estén siendo ejecutadas en ese momento dentro de otra unidad de procesado.

En la figura 1 se puede observar la arquitectura interna del micro con detalle representado en forma de esquema funcional.

• Velocidad:

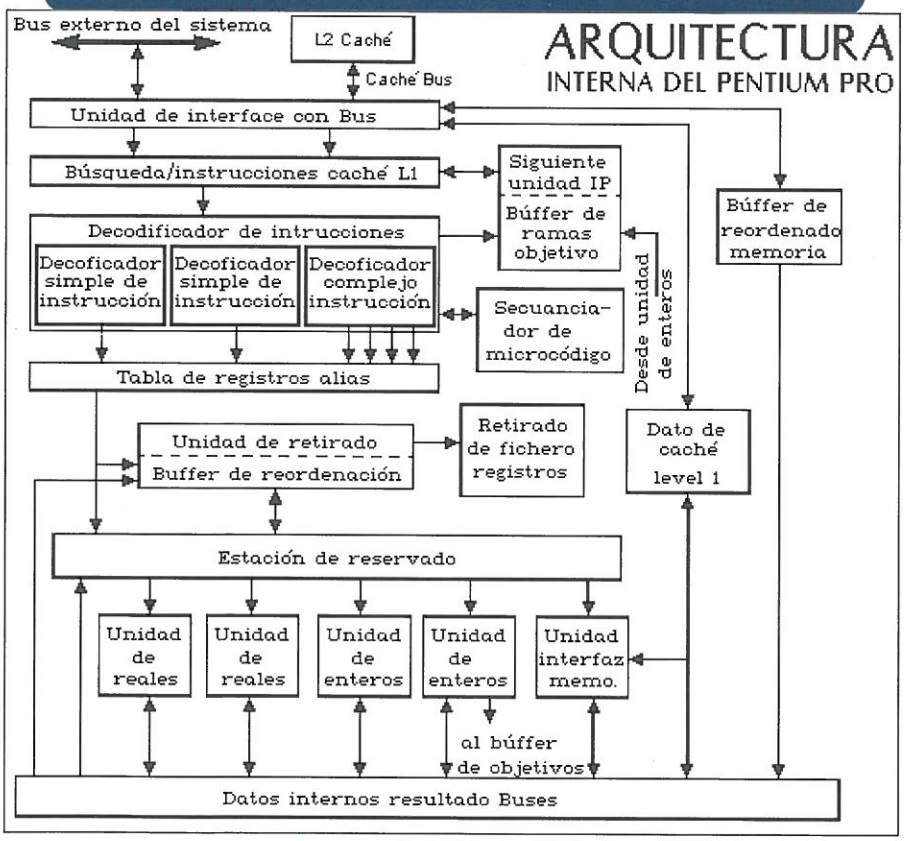
En la actualidad lo habitual es encontrar este micro a velocidades de 180 o 200 Mhz, pero también existen modelos a 150 y 166 Mhz y es casi seguro que aparecerán modelos a más de 200 Mhz en un futuro próximo.

• Memoria caché:

El Pentium Pro posee un total de 16 Kbytes de memoria caché level 1, que se encuentra dividida en dos subbloques de 8 Kbytes cada uno, los cuales están destinados al código y a los datos respectivamente.

En lo referente a la memoria caché level 2, tenemos que según el modelo de micro Pentium Pro del que estemos hablando, podemos encontrarlo en configuraciones de 256 o 512 Kbytes, siendo incluso posible, en el caso del modelo a 200 Mhz, elegir entre las dos cantidades, dependiendo de cuáles sean nuestras necesidades de rendimiento.

Figura 1. Esquema de la microarquitectura del Pentium Pro.



El Pentium Pro es totalmente compatible con toda la familia x86

La memoria caché level 2 que posee se encuentra, como hemos dicho, integrada también dentro del propio encapsulado del chip. La razón de ello es fácil de entender ya que gracias a este diseño ha sido posible conectar ambos, micro y caché level 2, con un bus de comunicación directo de escasa longitud (milímetros) que además de tener un ancho de banda real de 64 bits, funciona a la misma velocidad que el micro y no a la velocidad del bus externo (60 o 66). Con todo lo dicho, es fácil deducir que el rendimiento de esta caché se ha incrementado considerablemente.

• Consumo:

A una velocidad de 150 Mhz, consume 23 Watios.

• Instrucciones/funcionalidad:

Este micro es totalmente compatible con todos los micros de la familia x86 (o sea, puede ejecutar todo el soft existente en el mercado) y además incluye nuevas instrucciones y capacidades mejoradas y ampliadas en la gestión de memoria.

La dynamic execution

Esta nueva tecnología ha sido mencionada anteriormente. Su nombre se utiliza para designar globalmente tres conceptos de procesado de datos que son: predicción de saltos, análisis dinámico de datos y ejecución especulativa.

De ellos, la capacidad más importante es la tercera, que es la que hemos detallado anteriormente.

La *predicción de saltos* es una tecnología utilizada en microprocesadores de alta velocidad instalados en mainframes y consiste básicamente en que la CPU posee una especie de *caché de saltos* con los últimos cálculos de los saltos ejecutados dentro de nuestros programas, disminuyéndose así el tiempo que requiere la CPU para saltar a otro punto al que hayamos ido recientemente. Esto es muy útil en casos como bucles y algoritmos complejos, donde los saltos son continuos y normalmente reiterativos en cuanto a destinos.

El otro concepto que queda por definir es el de *análisis de datos dinámico*, que no es más que la capacidad que tiene la CPU de predecir los registros y datos que las instrucciones en curso requieren para poder así pasar a la ejecución en paralelo de instrucciones posteriores dentro de la ventana virtual de ejecución que no dependan en nada de los mismos.

Con ello, se hace posible que el microprocesador pueda realizar la ejecución especulativa.

Nuevas instrucciones del Pentium Pro

A continuación detallamos las nuevas instrucciones de que disponemos con este micro que no están presentes en los anteriores microprocesadores que pertenecen a la misma familia:

CMOVcc:

Ésta es una nueva familia completa de instrucciones, todas ellas destinadas a realizar un movimiento de datos condicional. Como es lógico, hay una variante de la misma para cada tipo de condición al igual que hay una instrucción de salto condicional para cada tipo.

En la tabla 1 se puede ver una lista completa de las instrucciones CMOVcc del Pentium Pro.

La transferencia se puede hacer desde memoria/registro hacia un registro y sólo permite operar con valores de 16 y 32 bits. Los datos de 8 bits no están soportados en ningún método.

El microcódigo que se realiza al ejecutarse es:

```
temp <- DEST
IF condition TRUE
THEN DEST <- SRC
ELSE DEST <- temp
FI;
```

FCMOVcc:

Esta instrucción es idéntica en cuanto a funcionamiento a las anteriores descritas, pero las banderas que se examinan son las equivalentes condicionales del coprocesador y los datos a mover siempre salen desde un registro del coprocesador hacia la cima de su pila.

Toda la familia de instrucciones FCMOVcc se puede obtener simplemente añadiendo una F como prefijo a todas las detalladas en la tabla 1.

FCOMI:

Esta instrucción, que también es relativa al coprocesador matemático (como todas las que empiezan por F), está diseñada para realizar la comparación de dos registros de la FPU y almacenar el resultado obtenido dentro las banderas de estado de la FPU en el registro EFLAGS de enteros. Hay cuatro variantes de esta instrucción, que son: FCOMI/FCOMIP y FUCOMI/FUCOMIP. Su utilidad queda patente, puesto que nos ahorrará muchas instrucciones expresas de traspaso de FLAGS entre unidades de procesado.

Pentium II es básicamente un Pentium Pro con tecnología MMX

RDPMC:

Mediante esta instrucción podemos leer el contenido de los registros de monitorización de la CPU.

Para poder controlar los permisos de lectura de estos registros, se ha incorporado el flag llamado PCE en el registro CR4 (localizado en el bit nº 8) que indica, cuando está activo, que se puede ejecutar esta instrucción desde cualquier programa desde nivel 0 hasta el nivel 3 de protección.

El microcódigo correspondiente a su ejecución es el siguiente:

```
IF (ECX = 0 OR 1) AND
        ((CR4.PCE = 1) OR
        ((CR4.PCE = 0) AND
         (CPL=0)))
  THEN EDX:EAX <- PMC[ECX];
ELSE (* ECX no es 0 o 1 y/o CR4.PCE es 0 y CPL
      es 1, 2, or 3 *) #GP(0)
FI;
```

UD2:

Esta instrucción, que significa literalmente <Undefined> instruction, simplemente sirve para generar una excepción de código.

Aunque parezca inútil, se ha incorporado al conjunto de instrucciones para poder comprobar los punteros de excepción de <*opcodes* inválidos>.

Instrucciones mejoradas y optimizaciones

A parte de las nuevas instrucciones, encontramos en este micro también algunas que han sido mejoradas en sus funcionalidades. Así, por ejemplo, las instrucciones RDMSR y WRMSR, que se encargan de la lectura y escritura de los registros monitorización y control *specific-mode*, han sido ampliadas en cuanto al número de tipos de eventos con los que pueden operar.

En lo referente a la velocidad de ejecución, el Pentium Pro ha sido optimizado globalmente en los tiempos de ejecución para poder ejecutar las instrucciones más rápidamente en modo 32 bits que en modo real, puesto que dicho micro ha sido diseñado con vistas al futuro inmediato, momento en que se usarán máquinas que se supone van a correr bajo Windows NT o similares (32 bits nativos).

Mejoras en la gestión de memoria

Otro apartado que se ha mejorado en muchos aspectos es la gestión de la memoria. A continuación damos detalles de lo que se ha añadido:

1.- Lo primero que nos encontramos y que da más vistosidad al micro, es que se ha ampliado el rango de direcciones físicas que puede gestionar el micro, disponiendo ahora de cuatro líneas más de direccionamiento que se activan mediante el bit nº 5 del registro CR4. Una vez puesto a uno dicho flag, disponemos de un total de 36 líneas que se traduce en la práctica en 64 Gigabytes direccionables, lo cual representa un sustancial aumento si tenemos en cuenta que un Pentium sólo llega a los 4 Gbytes.

Los registros de monitorización pueden controlar más de 100 eventos

2.- Gracias al bit nº 7 del registro CR4 llamado PGE, siglas que provienen del inglés *Page Global Enable* y pueden ser traducidas como Activación de páginas globales, se permite ahora marcar en una tabla las páginas de memoria más utilizadas para que no sean tenidas en cuenta al trasladar páginas, como ocurre en el caso de las páginas que contienen código del S.O., por ejemplo.

De esta forma, podemos conseguir una mejora en la velocidad de funcionamiento del equipo muy significativa.

3.- Con el bit nº4 del registro CR4, que se llama PSE (Extensión del tamaño de las páginas), ahora es posible utilizar páginas con tamaños permitidos de entre los 4KB y los 4 Mbytes.

Esto nos permite deducir que existe la posibilidad de definir páginas del doble de tamaño con respecto a los 4Kb/2Mb que anteriormente se permitían de forma habitual en los modelos anteriores de la familia Pentium.

Tabla 1. Instrucciones de transferencia condicional.

CMOVE/CMOVZ: Mover dato si "igual" o "cero"
CMOVNE/CMOVNZ: Mover dato si "no igual" o "diferente de cero".
CMOVA/CMOVNBE: Mover dato si "por encima" o "no inferior o igual".
CMOVAE/CMOVNB: Mover dato si "por encima o igual" o "no inferior"
CMOVB/CMOVNAE: Mover dato si "inferior" o "no superior ni igual".
CMOVBE/CMOVNA: Mover dato si "inferior o igual" o "no superior"
CMOVG/CMOVNLE: Mover dato si "mayor" o "no igual o menor"
CMOVGE/CMOVNL: Mover dato si "mayor o igual" o "no menor"
CMOVL/CMOVNGE: Mover dato si "menor" o no igual o mayor".
CMOVLE/CMOVNG: Mover dato si "igual o menor" o "no mayor".
CMOVC: Mover dato si hay "acarreo".
CMOVNC: Mover dato si "no hay acarreo".
CMOVO Mover dato si hay "desbordamiento".
CMOVNO: Mover dato si "no hay desbordamiento".
CMOVS: Mover dato si hay "signo".
CMOVNS: Mover dato si "no hay signo".
CMOVP/CMOVPE: Mover dato si "paridad" o si "paridad par".
CMOVNP/CMOVPO: Mover dato si "no paridad" o "paridad impar"

Los registros msr de monitorización y control hardware

En este micro se han incorporado dos registros que se usan como contadores de eventos y las funciones hardware que realiza la CPU como controladores. Estos registros no son nuevos de este Micro (aparecieron con el Pentium), pero sus ampliaciones en cuanto a posibilidades de funcionamiento han sido tan importantes que podemos considerarlos como tales.

Los eventos que ahora se pueden controlar son más de cien y ello es programable vía soft.

Ejemplos de contadores serían: el número de instrucciones decodificadas hasta el momento, las interrupciones recibidas, las veces que se ha cargado la caché, etc.

Estos contadores se inicializan, arrancan y paran usando las instrucciones RDMSR y WRMSR junto con la nueva instrucción especifica del Pentium Pro RDPMC.

Los registros MTRR

Los registros MTRR son una nueva capacidad que se ha incorporado a los Pentium Pro que está destinada a optimizar las operaciones de entrada y salida de memoria para los diferentes tipos con que puede encontrarse el micro instalados en el sistema (ROM, RAM, memoria mapeada I/O, etc.)

Los MTRR's no son en realidad más que registros MSR que permiten configurar mapas de rangos de memoria con cada uno de los tipos que pueden contener.

Figura 2. Tabla con los valores obtenidos con CPUID según el micro.

DATOS OBTENIDOS EN ‹EDX› TRAS EJECUTAR ‹CPUID›

TIPO	FAMILIA	MODELO	STEPPING	DESCRIPCION DEL MICROPROCESADOR
00	0100	000(0/1)	*1	Intel 486DX.
00	0100	0010	*1	Intel 486SX.
00	0100	0011	*1	Intel 487 CPU.
00	0100	0011	*1	Intel DX2.
00	0100	0011	*1	Intel DX2 Overdrive.
00	0100	0100	*3	Intel 486 SL.
00	0100	0101	*1	Intel SX2.
00	0100	0111	*3	Intel DX2 mejorado (Write-back)
00	0100	1000	*3	Intel DX4.
00/01	0100	1000	*3	Intel DX4 OverDrive.
00	0101	0001	*2	Pentium 60/66 Mhz.
00	0101	0010	*2	Pentium 75/90/100/120/133/150/166/200 Mhz.
00	0101	0001	*2	Pentium Overdrive para Pentium 60/66 Mhz.
00	0101	0010	*2	Pentium Overdrive para Pentium 75-133 Mhz.
01	0101	0011	*2	Pentium Overdrive para 486.
00	0101	0100	*2	Pentium MMX 166/200 Mhz.
01	0101	0100	*2	Futuro Pentium MMX Overdrive para P75 Mhz.
00	0110	0001	*2	Pentium Pro.
00	0110	0011	*2	P6 MMX (Pentium II).
01	0110	0011	X	Futuros Overdrive para Pentium Pro

*1=CPUID no soportado *2=Varia según modelos *3= Acepta la instrucción CPUID

Estas tablas son usadas internamente por el microprocesador para poder saber a qué clase de memoria está accediendo en cada momento y poder así determinar si se debe usar la caché o no para ello y qué sistema de acceso es el más óptimo para la operación de entrada/salida.

Con este sistema se pueden indicar cinco tipos de memoria que el microprocesador usará de diferentes formas. Éstos son los siguientes: UC (*UnCacheable*) que es la no cacheable, WC (*Write-Combining*) que es la de escritura combinada, la famosa WT (*Write-Trought*) de escritura única, la WP (*Write-Protected*) de escritura protegida y la WB (*Write-Back*) que es la de escritura "diferida".

Todo ello, además de mejorar el rendimiento del sistema, tiene como consecuencia añadida que el diseño del micro se simplifique a nivel físico, debido sobre todo a que la patilla #KEN y la lógica externa de mapeado utilizada en los microprocesadores 486 y Pentiums no son en absoluto necesarias.

Detección del Pentium Pro

Como sabemos, a partir del Pentium disponemos de la llamada instrucción CPUID, la cual nos devuelve toda la información necesaria acerca del micro instalado (lo cual ya devolvía el 486 pero sólo tras un reset).

En la figura 2 podemos ver todos los valores que devuelve la instrucción CPUID en el registro general EDX y sus significados correspondientes.

Para saber si el micro sobre el cual está ejecutándose el programa soporta la instrucción CPUID, intentaremos cambiar el bit nº 21 del registro EFLAGS. Si lo conseguimos, entonces la instrucción CPUID se puede ejecutar y podremos usarla.

Dentro del contenido devuelto en el registro EDX por CPUID, la información está dividida en varias zonas, que contienen datos del Tipo (bits 16-17), *Modelo*

(bits 12-15), *Familia* (bits 8-11) y el llamado *Stepping* (bits 0-7). Como se puede observar, los bits 18-31 quedan reservados para su uso en futuros procesadores.

Los dos bits reservados al apartado Tipo, que pueden representar cuatro combinaciones, identifican lo siguiente: para <00> será Procesador OEM original, con el valor <01> significa Procesador Overdrive, valor <10> significa Procesador Dual y el valor <11> está reservado para el futuro por Intel.

Este micro amplia a 64 Gb la capacidad de direccionamiento

Si lo de procesador Dual le suena raro, recuerde que estos micros han sido diseñados con capacidades de proceso simétrico para sistemas bi-procesadores.

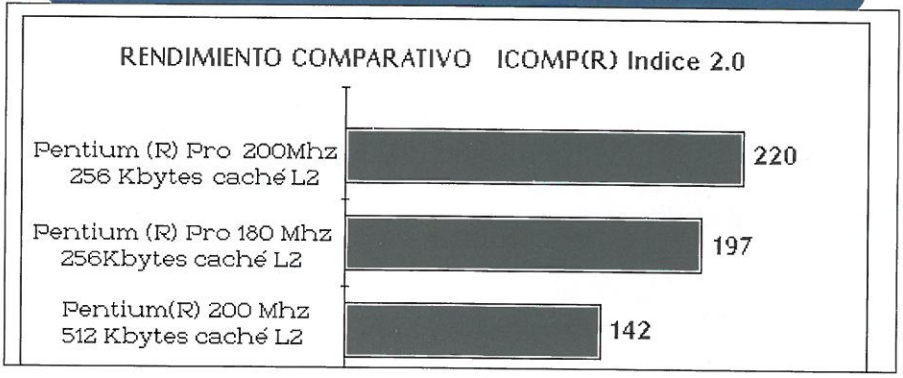
Rendimiento del Pentium Pro

Como ya se ha mencionado, el Pentium Pro, gracias a su nueva tecnología de diseño, permite ejecutar más instrucciones por segundo y más en paralelo. Ello, junto a las optimizaciones que se le han aplicado para correr más rápidamente sólo en modo 32 bits (Windows NT), hace que los programas más modernos se beneficien claramente de él.

El Pro Posee dos unidades de enteros, dos de FPU y una de memoria

También hay que tener presente que si va a usarlo como base de un sistema operativo como Windows 3.1 o DOS, las mejoras van a ser muy inferiores.

Figura 3. Comparativa ICOMP(R) 2.0 entre Pentium y Pentium Pro.


RENDIMIENTO COMPARATIVO ICOMP(R) Indice 2.0
Pentium (R) Pro 200Mhz 256 Kbytes caché L2
220
Pentium (R) Pro 180 Mhz 256Kbytes caché L2
197
Pentium(R) 200 Mhz 512 Kbytes caché L2
142

En la figura 3 se puede comparar el rendimiento de tres equipos con el Índice ICOM 2.0. Como se puede observar, el Pentium Pro ofrece un rendimiento del doble sobre el de un Pentium a idéntica velocidad de reloj con el doble de memoria caché level 2 instalada.

Para la realización de esta prueba se usaron en los dos equipos Pentium Pro la siguiente configuración: *ChipSet 440FX, 64 Mbytes* de EDO/ECC, Controladora de disco duro *Adaptec 2940UW SCSI/PCI*, disco duro *Seagate ST31* y tarjeta gráfica *Matrox Millenium PCI* con los drivers 2.10 con un modo gráfico de 1024*768*256 colores.

En el caso del Pentium, se usó una configuración idéntica pero con el Chipset 430VX en la placa base.

Programación optimizada

Explicada toda la arquitectura interna de este micro, queda claro que con el sistema de Ejecución Dinámica de instrucciones con el que está diseñado, si queremos realizar código nativo para él que aproveche al 100% las capacidades del sistema, además de aplicar las normas típicas de optimización ensamblador, debemos intentar crear algoritmos que realicen operaciones diferentes de forma paralela, o sea, que aprovechen las capacidades de procesado en paralelo y especulativo del micro escribiendo el código con operaciones sobre diferentes registros y

El mayor avance incorporado es la llamada "ejecución especulativa"

zonas de memoria de forma intercalada, así, el tiempo durante el cual las unidades de enteros y reales quedan inactivas, será mínimo.

Una muestra de ello sería, por ejemplo, si tuviésemos que calcular algo sobre dos valores dados, que podrían estar contenidos en EAX y EBX. Si nos fuese posible, tendríamos que realizar las operaciones sobre dichos registros escribiendo líneas de código manipulando EAX y EBX alternativamente. El código resultante será más rápido que el equivalente con los dos algoritmos separados y colocados secuencialmente y su eficacia será más notable todavía en el caso de que estemos hablando de un Pentium normal, que dispone de capacidad de procesado en paralelo pero no de ejecución especulativa, que sólo aparece a partir del Pro. La única desventaja de este método es que la depuración del código resulta más compleja y es más fácil cometer errores.

LINUX

Programación X-Window: Driver SVGA para XFree86

Jorge F. Delgado Mendoza

Uno de los mayores problemas con los que se topa la gente al instalar Linux, u otro Unix de distribución gratuita, en sus PC's es que su nueva tarjeta de vídeo no esté soportada por el entorno de Ventanas X-Window. Daremos unas nociones de arquitectura de SVGA's y describiremos los pasos para construir un driver de vídeo básico, integrándolo en XFree86.

Ya has conseguido instalar tu última versión de *Linux*, has recompilado el *kernel* del sistema para incluir tus distintos periféricos y te dispones a configurar las X-Window, el entorno de ventanas *Unix*. Siguiendo todos los pasos del README adecuado ejecutas el programa *SuperProbe*, llevándote tu primera sorpresa al descubrir que lo único que detecta es una tarjeta MCGA o VGA.

Tras muchos intentos, consigues configurar un servidor de 16 colores, pero has descubierto que la tarjeta de vídeo maravillosa que te acabas de comprar no está soportada por el sistema de ventanas, es horriblemente lenta y, desde luego, trabajar en blanco y negro o con 16 colores, no es lo que esperabas de tu sistema operativo favorito.

Al fin y al cabo, dejando de lado la velocidad, ¿que diferencia hay entre una SVGA y otra?, ¿acaso no son todas compatibles con el estándar VGA? La respuesta es sencilla: sí, son compatibles VGA, pero a partir de ahí todas son distintas. La razón es que el estándar VGA solo definía resoluciones de hasta 320x200 en 8 bpp, y de 640x480 en 4 bpp[1]. Más allá tenemos una colección de SVGA's todas diferentes.

El verdadero problema reside en que, cuando se quisieron obtener mayores resoluciones, cada fabricante de tarjetas de vídeo decidió hacer la guerra por su cuenta. Aunque manteniendo siempre compatibilidad con el estándar VGA, cada uno implementó su propio sistema de conseguir modos de mayor resolución, capaces de presentar más cantidad de colores en la pantalla. Este problema, que al principio se limitó al uso de distintos *I/O ports*, - puertos de entrada o salida de datos - para controlar la tarjeta, se agudizó aun más con la llegada al mercado de los aceleradores. Hoy en día cada tarjeta emplea un coprocesador gráfico diferente, siendo la mayoría de las veces incompatibles incluso entre distintos modelos del mismo fabricante.

Son estos motivos los que obligan a la escritura de un nuevo *driver* cada vez que aparece un nuevo *chipset* - coprocesador gráfico de la tarjeta - en el mercado. Normalmente las compañías desarrollan los programas encargados de proporcionar un conjunto de funciones gráficas básicas, es decir los *drivers*. Sin embargo se limitan a soportar Windows95, WindowsNT y algunas incluyen soporte para el estándar VESA en la BIOS de la propia tarjeta o como TSR, de tal modo que en MS-DOS los juegos puedan usar todos los modos de resolución disponibles. Los usuarios de *Linux* nos encontramos, pues, en desventaja ya que ¿quién hace los *drivers* para las nuevas tarjetas?

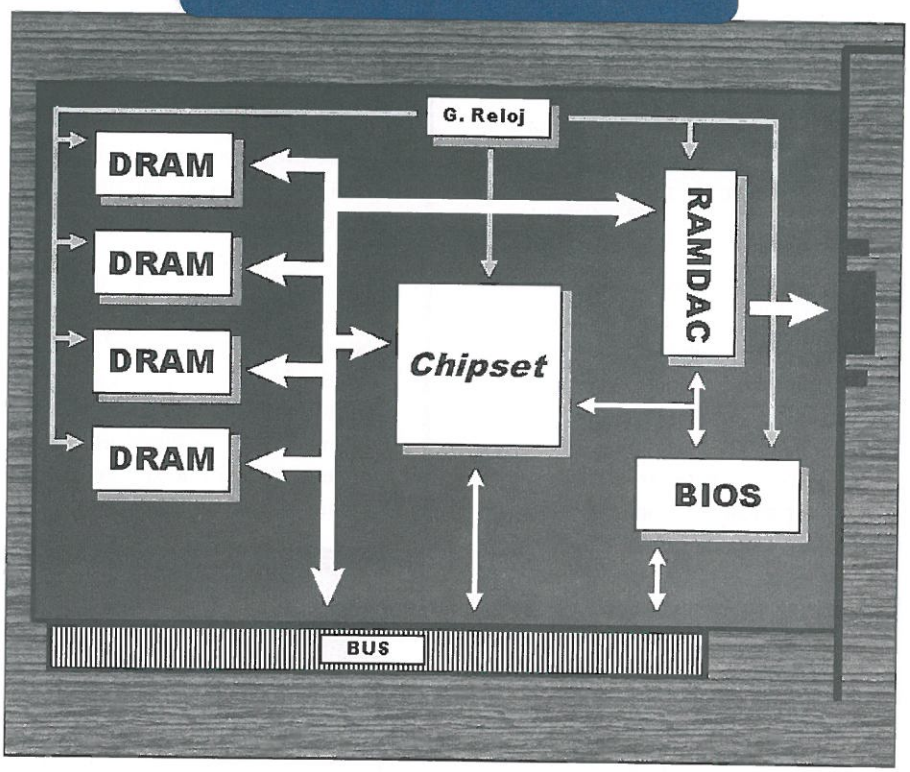
De codificar los *drivers* se encarga el *XFree86 Project Inc.*, una organización no lucrativa que tiene como propósito proporcionar a la comunidad de usuarios de los distintos *Unix* que existen actualmente en el mercado un sistema X-Window

moderno, rápido y fiable, además de gratuito. Para ello, todos los componentes del grupo, desde los miembros del Cuadro de Directores hasta el último *beta tester*, colaboran de manera gratuita y desinteresada en su tiempo libre. Por supuesto no es suficiente; siempre hay alguna tarjeta que queda sin soportar. Entre los motivos podemos encontrar la falta de una placa sobre la que probar el código, la carencia de documentación o, simplemente, la falta de tiempo.

A lo largo de este artículo se van a describir los pasos fundamentales para la construcción de un *driver* para una tarjeta de vídeo genérica. Sin embargo, antes de empezar, es necesario hacer un breve comentario: la realización de un *driver* de tarjeta de vídeo es una tarea compleja, desagradecida y, muchas veces, frustrante. Si no se tiene cuidado se corre el riesgo de dañar el monitor y, desde luego, cualquier error en la codificación es probable que cuelgue la máquina de tal modo que sólo el *Reset* sirva para recuperar el control de la misma. Sin embargo, cuando por fin el servidor arranca y funciona, la satisfacción que se obtiene es inmensa.

En cuanto al riesgo que comporta, las precauciones que hay que tomar son básicas. Evitar llevar al monitor más allá de sus especificaciones es la principal. Lo mismo se puede decir de los componentes de la tarjeta de vídeo. Al estar programando directamente un dispositivo, conviene tener muy claro qué es lo que se quiere conseguir y cómo. El método de ensayo y error no es apropiado ni aconsejable.

Figura 1. Componentes principales de una tarjeta de vídeo e interrelación entre ellos. Las líneas gruesas corresponden a flujos de datos, las finas a flujos de control.



Requisitos

Conocimientos: la mayor parte del *driver* se va a construir utilizando código C, por tanto es necesario dominar este lenguaje de programación. Además, es necesario realizar algunas rutinas en código máquina, por lo que también es conveniente tener algunas nociones de ensamblador para x86. Finalmente, es preciso saber qué es cada componente en una tarjeta de vídeo y conocer las interacciones entre ellos, de lo cual nos vamos a encargar en el presente artículo.

Software: necesitaremos tener instaladas las siguientes herramientas:
- *gcc* 2.7.0 o superior.
- *make*.
- *XFree86*.
- *LinkKit*.

Documentación: es necesario un manual con la descripción, a nivel de registro, de la tarjeta de vídeo para la que queramos añadir el *driver*. Muchas compañías dan estos manuales de modo gratuito. Otras cobran una pequeña cantidad y, muy pocas, exigen la firma de un NDA[2]. Si nos es imposible conseguir un manual, el programa *freeware* VGADOC, contiene la descripción de un gran número de tarjetas.

Además, conviene tener a mano el manual del monitor con las especificaciones de refresco, tanto vertical como horizontal, así como su máxima frecuencia de punto (*Dot Clock*). **Nunca** deberemos rebasar estos valores.

(1) A partir de ahora nos referiremos a la cantidad de colores de un modo de vídeo por el número de *bits per pixel* u octetos por punto. Así, 16 colores serán 4 bpp, 256 serán 8 bpp, 65536 colores serán 16 bpp y 16.7 millones de colores serán 24 bpp o 32 bpp, reflejando cuántos bits ocupa cada punto en la memoria de la tarjeta de vídeo.

(2) NDA: *Non Disclosure Agreement*. Documento según el cual el firmante se compromete a no revelar la información recibida de ningún modo. Ni como documentación, ni como código fuente.

Arquitectura de una tarjeta de vídeo

En una tarjeta de vídeo podemos distinguir seis componentes principales, estrechamente relacionados entre sí. Estos componentes son: la memoria o *frame buffer*, el Generador de Reloj, la BIOS VGA, el *bus* de acceso, el *chipset* y el DAC.

Generador de Reloj: es el encargado de generar y transmitir las señales que van a gobernar el funcionamiento de los distintos componentes de la tarjeta. Puede ser programable o tener una serie de señales prefijadas a distintas frecuencias. Su importancia a la hora de realizar un *driver* es la de ser el encargado de generar las diferentes frecuencias necesarias para controlar los distintos modos de vídeo y la señal *MCLK*, que controla el acceso a la memoria de la tarjeta.

Bus de Acceso: es el camino por el que se transmiten los datos del generador en el ordenador a la Tarjeta de Vídeo y viceversa. Puede ser un *bus* ISA, VLB, EISA o bien PCI. La velocidad del *bus* condiciona el rendimiento de la tarjeta, especialmente si no se utilizan las funciones aceleradas del *chipset*.

VGA BIOS: del inglés *Vídeo Graphics Array Basic Input Output Service*, proporciona la funcionalidad básica VGA de la tarjeta. En las placas más modernas lleva incorporadas las funciones del estándar VESA.

RAMDAC: de nuevo un acrónimo obtenido del inglés. Es un conversor digital/analógico encargado de transformar los bits de información presentes en la memoria de la tarjeta de vídeo en señales analógicas susceptibles de ser entendidas por un monitor multifrecuencia.

En los modelos más recientes de placas, el DAC y el Generador de Reloj pueden ser un único componente denominado GENDAC. La máxima frecuencia a la que el DAC puede operar condiciona, junto con la cantidad de memoria instalada, la máxima resolución que puede obtenerse con una Tarjeta determinada. Finalmente, la máxima frecuencia de refresco que puede alcanzarse para una resolución dada también viene determinada por el DAC de la tarjeta.

Los límites de refresco suelen ser superiores a los del monitor, a no ser que éste sea de alta calidad (generalmente de 17" o más). Este es el punto en el que más cuidado se tiene que tener, evitando utilizar un monitor más allá de sus frecuencias nominales.

Memoria o Frame Buffer: en ella se almacena la imagen que ha de aparecer en el monitor en forma de una secuencia de bits. La cantidad de memoria disponible condiciona la resolución máxima que se puede alcanzar, siempre que se tenga un RAMDAC suficientemente rápido, según la Tabla 1.

Para acceder a esta memoria desde un programa, se utiliza el mismo método que para acceder a la memoria principal de un ordenador: lecturas y escrituras a direcciones de memoria. Existen dos métodos para realizar esta operación, el direccionamiento segmentado y el direccionamiento lineal (*segmented* y *linear addressing modes*):

- **Segmentado:** presente en todas las tarjetas compatibles VGA.

Tabla 1.- Máximas resoluciones alcanzables para un cantidad de memoria dada. Todos los modos suponen que el RAMDAC es capaz de soportar el modo gráfico correspondiente.

Modo/Profundidad	Memoria	512 Kbytes	1 Mbyte	2Mbytes	4Mbytes
640x480	4bpp	Sí	Sí	Sí	Sí
800x600	4bpp	Sí	Sí	Sí	Sí
640x480	8bpp	Sí	Sí	Sí	Sí
800x600	8bpp	Sí	Sí	Sí	Sí
1024x768	8bpp	No	Sí	Sí	Sí
1152x900	8bpp	No	Sí	Sí	Sí
1280x1024	8bpp	No	No	Sí	Sí
1600x1280	8bpp	No	No	Sí	Sí
640x480	16bpp	No	Sí	Sí	Sí
800x600	16bpp	No	Sí	Sí	Sí
1024x768	16bpp	No	No	Sí	Sí
1152x900	16bpp	No	No	Sí	Sí
1280x1024	16bpp	No	No	No	Sí
1600x1024	16bpp	No	No	No	Sí
640x480	24bpp	No	Sí	Sí	Sí
800x600	24bpp	No	No	Sí	Sí
1024x768	24bpp	No	No	No	Sí
1280x1024	24bpp	No	No	No	Sí
1600x1280	24bpp	No	No	No	No
640x480	32bpp	No	No	Sí	Sí
800x600	32bpp	No	No	Sí	Sí
1024x768	32bpp	No	No	No	Sí
1280x1024	32bpp	No	No	No	No

Originalmente el estándar sólo contemplaba la presencia de 64 Kbytes de RAM en la tarjeta, por lo que se reservó este espacio en el mapa de memoria de un PC en la posición 0xA0000.

Evidentemente, en los modos superiores que requieren más memoria sólo podríamos acceder a una pequeña parte de la pantalla, por lo que los fabricantes decidieron utilizar este espacio de 64 Kbytes (0xA0000 - 0xAFFFF) como una *ventana* a la cantidad total de memoria. Organizando la memoria en bloques de 64 Kbytes y con un registro extra que indique el número de banco, se puede acceder a toda la memoria de la tarjeta. (Aquí surgió la primera de las incompatibilidades, ya que cada fabricante utilizó un sistema de organización diferente y un *I/O Port* distinto para acceder a los registros).

• **Lineal:** surgió como mejora del modo segmentado y está solo presente en las tarjetas más modernas. La solución consiste en considerar la memoria como un solo bloque y mandarla a una posición muy alta en el mapa de memoria del PC.

Inicialmente se utilizó 0xE00000 (el Megabyte 14), pero con el aumento de memoria en los PCs actualmente se utilizan posiciones en el orden de los Gigabytes. Esta solución mejora las prestaciones de la tarjeta - entre otras cosas al no requerir rutinas de cambio de banco -, sin embargo vuelve a no pertenecer al estándar VGA y necesita la creación de *software* ad hoc para utilizarlo.

Otro detalle a tener en cuenta es que el ancho de banda de la memoria, es decir, el número de *bytes* que pueden escribirse en ella por unidad de tiempo, condiciona la velocidad del acelerador.

En los modernos aceleradores de 64 bits se suele obtener una gran mejora en prestaciones al añadir el segundo *Megabyte*, ya que se usa memoria de 32-bits organizada en dos bancos. Si sólo tenemos instalado un *Megabyte*, para escribir 64 bits necesitaremos tres ciclos de memoria en lugar de un único acceso, debido a lo cual el ancho de banda es, aproximadamente, un tercio.

• **Chipset o Coprocesador Gráfico:** es el corazón de la tarjeta, encargado de controlar todas sus funciones. Su parte más importante desde nuestro punto de vista podría ser el *Graphic Engine* o acelerador gráfico, por lo que cuando nos refiramos al *chipset* a lo largo de este artículo, estaremos haciendo referencia al acelerador propiamente dicho. Su función es la de implementar rutinas *hardware* de dibujo 2D y 3D, con un doble fin:

• Aliviar carga de la CPU principal del sistema realizando tareas complejas en base a unos pocos parámetros.

• Aprovechar la proximidad arquitectural al *frame buffer* y los demás componentes, con el fin último de que las funciones complejas que éste realiza se completen en un tiempo menor que si fuera la CPU principal del sistema la que las llevara a cabo.

El número, la calidad y la velocidad de las funciones que implemente el acelerador van a ser las que marquen la velocidad máxima que puede alcanzar una tarjeta.

Desafortunadamente, cada *chipset* implementa un número diferente e incompatible de éstas.

La desventaja de esta arquitectura reside en la necesidad de utilizar un programa que haga de interfaz entre las aplicaciones y el acelerador. Este programa, el *driver*, puede implementar todas, algunas, o ninguna de las funciones *hardware* del acelerador.

Éste, y no otro, es el motivo de la gran importancia de tener *drivers* actualizados para una tarjeta, ya que el rendimiento real de la misma viene condicionado por el número de funciones que ofrezca el *driver*.

Primer paso: configurar el sistema

El primer paso para comenzar a construir el código del *driver* es conocer el entorno en el que nos vamos a mover, el *LinkKit*. Concebido inicialmente para enlazar servidores ya construidos, actualmente permite añadir un nuevo *driver* al entorno de ventanas, sin necesidad de tener instaladas todas las fuentes de X-Windows.

Está instalado normalmente en el directorio**/usr/X11R6/lib/Server.**

Dentro del *LinkKit* podemos encontrar, entre otros, los siguiente elementos:

• **XF86SITE.DEF** Configuración general del *LinkKit*.
• **include/** Cabeceras y macros necesarias para la compilación.
• **xf86.h** Contiene la estructura *ScrnInfoRec*.
• **xf86_Option.h** Todas las opciones que queramos pasar al servidor deberán estar incluidas aquí en tiempo de compilación.
• **compiler.h** Compendio de macros, utilidades y código ensamblador útiles para la construcción del servidor.
• **vga256.h** Rutinas de dibujo en 256 colores (8bpp).
• **drivers/** Los *drivers* de los diferentes *chipsets* se encuentran en este directorio.
• **vga256/** *Drivers* de 256 colores. Aquí podemos encontrar numerosos ejemplos que serán muy útiles. Por su sencillez de comprensión quizá destaque el driver de Oak Tech Inc.
• **vga16/** *Drivers* de 16 colores.
• **vga2/** *Drivers* monocromo.
• **VGADriverDoc/** En este directorio podemos encontrar documentación, aunque un poco anticuada, sobre la construcción de *drivers* y, lo más importante, los prototipos SVGA que vamos a utilizar como modelo para crear el nuestro.

- **Imakefile.stub** El *Imakefile* que vamos a utilizar para compilar el *driver*.
- **stub_bank.s** El prototipo de rutina ensamblador de cambio de banco.
- **stub_driver.c** El prototipo de *driver* escrito en código C.

Desde aquí y hasta el final del artículo, todos los *paths* mencionados se darán, a no ser que se indique lo contrario, a partir del directorio **/usr/X11R6/lib/Server/**.

El número de bytes por unidad de tiempo, condiciona la velocidad

Una vez nos hayamos familiarizado con la estructura de directorios, estaremos en condiciones de comenzar a crear código. Lo primero que debemos hacer es confirmar que las herramientas *make* y *gcc* están en el sistema. Para ello teclearemos las siguientes líneas en la línea de comandos:

```
gcc <ENTER>
make <ENTER>
```

Debiéndose obtener, respectivamente:

```
gcc: No input files
make: *** No targets specified and no makefile
      found. Stop.
```

Si obtenemos las respuestas indicadas, estamos en condiciones de comenzar a trabajar. Lo siguiente es crear el directorio donde vayamos a poner todo nuestro código. La tarjeta que vamos a utilizar de ejemplo se va a llamar *Graphics'r'Us*, así que llamaremos *gu* al directorio que creemos. Este directorio debe estar situado en el siguiente punto de la jerarquía: **drivers/vga256/**

Una vez lo hayamos creado, es el momento de copiar dentro de él los prototipos de *driver* que encontraremos en **VGADriverDoc/**. Además, deberemos personalizarlos para nuestra tarjeta, cuyas características fundamentales se pueden observar en la Figura 2. Los renombraremos según el siguiente modelo:

```
stub_bank.s      ------> gu_bank.s
stub_driver.c    ------> gu_driver.c
Imakefile.stub   ------> Imakefile
```

Una vez renombrados, editaremos cada uno de ellos individualmente, reemplazando por *"gu"* todas las referencias a *"stub"* que encontremos en los ficheros. Por ejemplo, en el fichero *Imakefile*, nos encontramos las siguientes líneas:

```
SRCS = stub_driver.c stub_bank.s
OBJS = stub_driver.o stub_bank.o
```

Tras el cambio, las líneas deberían presentar el siguiente aspecto:

```
SRCS = gu_driver.c gu_bank.s
OBJS = gu_driver.o gu_bank.o
```

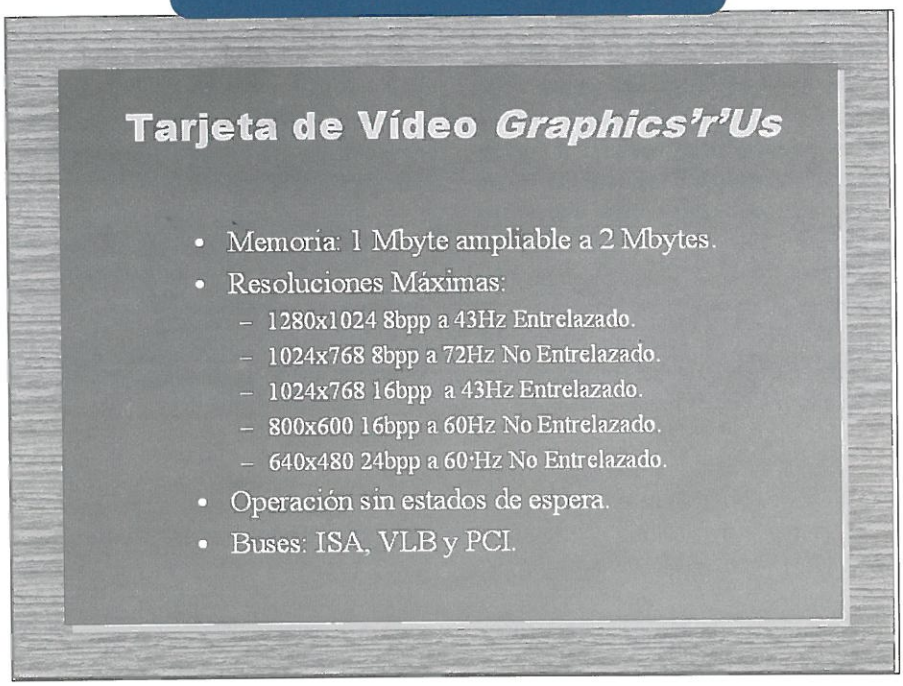
De la misma manera modificaremos el fichero *gu_driver.c*. Al hacer estos cambios conviene tener cuidado, ya que, aunque los fuentes de C no lo sean tanto, los *(I)makefiles* suelen ser bastante quisquillosos en cuanto a la falta o exceso de un espacio o un tabulador. Si a la hora de compilar obtenemos un error en los *Makefiles* lo ideal es volver a copiar el *Imakefile.stub* en el directorio de trabajo y realizar de nuevo la sustitución con más cuidado.

Una vez preparados los ficheros que compondrán el *driver*, hemos de añadir éstos al *LinkKit*, de tal manera que los incluya en el árbol de directorios de compilación. Para ello editaremos el fichero *XF86Site.def* añadiendo el *chipset "gu"* a la lista de los enumerados para el *driver* SVGA. Añadiremos el nuestro justo antes del *driver* genérico VGA, para evitar que nuestra rutina de detección interfiera con las ya existentes. El cambio se realizará de la siguiente manera:

```
#define XF86SvgaDrivers nv
        et3000 et4000 oak ... .... .... \
        cirrus ... ... ... ... ... ... ... \
        generic
```

Quedará sustituido por lo siguiente:

Figura 2. Características de la Tarjeta de Vídeo utilizada como ejemplo. Marca Sólo Programadores Tech Inc, con chipset Graphics'r'Us.


Tarjeta de Vídeo Graphics'r'Us
• Memoria: 1 Mbyte ampliable a 2 Mbytes.
• Resoluciones Máximas:
– 1280x1024 8bpp a 43Hz Entrelazado.
– 1024x768 8bpp a 72Hz No Entrelazado.
– 1024x768 16bpp a 43Hz Entrelazado.
– 800x600 16bpp a 60Hz No Entrelazado.
– 640x480 24bpp a 60 Hz No Entrelazado.
• Operación sin estados de espera.
• Buses: ISA, VLB y PCI.

```
#define XF86SvgaDrivers          nv
        et3000 et4000 oak ... .... .... \
        cirrus ... ... ... ... ... ... ... ... \
        gu generic
```

Una vez realizadas estas modificaciones, el entorno estará preparado para trabajar en la codificación del *driver*.

Segundo Paso: Codificación de las Rutinas de Cambio de Banco

Después de documentarnos sobre la estructura de una tarjeta de vídeo y de preparar el entorno sobre el que vamos a trabajar, ha llegado el momento de construir el *driver*. El objetivo del mismo será el de añadir soporte básico, modos de 8 bpp sin acelerar, en el servidor SVGA para una tarjeta imaginaria. Ésta tendrá una arquitectura lo suficientemente sencilla como para que sirva de ejemplo, sin perder generalidad. La Figura 3 describe la interfaz extendida de la tarjeta de vídeo ejemplo *Graphics'r'Us*. Una información más detallada, necesaria para la realización del *driver,* se encuentra en el CDROM de la revista (DataBook.ps).

Como ya se ha comentado, el estándar VGA solo contempla 64 Kbytes de memoria de vídeo, colocados en la posición 0xA0000. Para poder acceder a cantidades de memoria superiores a 64 Kbytes se utiliza un sistema para multiplexar toda la memoria de la tarjeta de vídeo en esta posición, utilizando las direcciones 0xA0000 a 0xAFFFF como si de una o varias ventanas se tratase. Cada una de estas porciones de memoria, de tamaño igual o menor a 64 Kbytes, se conoce por el nombre de *banco*. Existen tres posibles organizaciones de estos bancos:

Bancos Independientes: utilizan un registro para indicar el banco de lectura y otro para el de escritura. De esta manera podemos tener dos accesos en diferentes posiciones de la tarjeta, por lo que para trasladar información de un lado a otro bastará con un **mov**. Ésta es la organización ideal y es la que usa la tarjeta *Graphics'r'Us*.

Bancos Simultáneos: de nuevo utilizan dos registros pero, a diferencia del método anterior, ambas ventanas son de lectura/escritura. Evidentemente el tamaño de ambas ventanas deberá sumar 64 Kbytes. Esta organización es más compleja, pero muy interesante para operaciones de intercambio de datos.

Banco Único: es la más antigua y menos útil forma de acceso. Posee un registro único para la ventana, la cual es de lectura/escritura. De este modo, si deseamos mover datos entre dos posiciones de la tarjeta situadas en diferente banco, en vez de una operación **mov**, nos veremos obligados a hacer una combinación **load/store/write**, con una variable intermedia, lo cual es *mucho* más lento.

Debido a lo crítico de estas operaciones en el rendimiento de la tarjeta, las rutinas de cambio de banco se realizan en ensamblador, encontrándose en **gu_bank.s**

La sintaxis ensamblador que se ha de usar, por razones de portabilidad, está incluida en el fichero: **include/assyntax.h.**

Utilizando este macro-ensamblador podremos reutilizar el código escrito en varios sistemas operativos, - *Linux*, *NetBSD*, *FreeBSD*, *Solaris*, *SCO Unix*, etc.., - sin necesidad de realizar una rutina de cambio de banco para cada uno de ellos, ya que cada cual tiene una implementación diferente de las primitivas ensamblador.

Deberemos rellenar o recodificar las tres funciones presentes en el fichero, teniendo en cuenta que el banco al que se quiere conmutar, tanto si es una operación de lectura como de escritura, se nos pasa en el registro AL (8 bits menos significativos del acumulador de 32 bits EAX):

GUSetRead(): esta función selecciona a qué banco va a apuntar la ventana de lectura. Por tanto, bas-

Figura 3. Listado de los registros extendidos de la tarjeta de vídeo GU (Graphics'r' Us) utilizada como ejemplo.

Registro	R/W	Puerto	Índice	Bits
Extensión	R/W	0x3DE	N/A	8
Identificación	R	0x3DF	0	8
Configuración	R	0x3DF	1	8
Estados de Espera	R/W	0x3DF	2	8
Selector de Reloj	R/W	0x3DF	3	4
Overflow	R/W	0x3DF	4	4
HSYNC	R/W	0x3DF	5	8
CRTC Extendido	R/W	0x3DF	6	3
Profundidad de la *FIFO*	R/W	0x3DF	7	4
Selector de Lectura	R/W	0x3DF	8	5
Selector de Escritura	R/W	0x3DF	9	5

tará con que introduzcamos el valor de AL en el registro *Selector de Lectura*, índice 8 del manual de la tarjeta.

GUSetWrite(): de manera similar a la anterior, esta función solo modifica el número del banco situado bajo la ventana de escritura. Deberemos introducir el valor de AL en el *Selector de Escritura*, índice 9.

GUSetReadWrite(): con esta función colocamos el mismo banco simultáneamente bajo la ventana de lectura y la de escritura. Deberemos introducir el valor en ambos índices 8 y 9.

En el código presentado como ejemplo en el CDROM, hemos realizado estas funciones mediante la optimización descrita en la figura 4. Este método nos permite ahorrar una operación E/S, las cuales consumen una gran cantidad de ciclos de reloj.

Tercer Paso: Escribir el Driver

Una vez tenemos una rutina de cambio de banco escrita, llegó el momento de la tarea más compleja: la escritura del *driver* en **gu_driver.c**. Para ello, debemos tener en cuenta que un *driver* ha de cumplir las siguientes funciones dentro de la implementación de XFree86:

Crear una estructura de datos referente al *chipset* de tal manera que el servidor de X-Windows conozca las capacidades de la tarjeta. Esta estructura contiene punteros a ciertas funciones clave, así como algunos datos de la arquitectura de la tarjeta.

Almacenar los valores de los registros extendidos de la tarjeta para los distintos modos. Para ello crearemos otra estructura de datos local al *driver*.

Obtener la identificación del *chipset* o *chipsets* que soporta el *driver*.

Realizar la autodetección del *chipset* devolviendo verdadero o falso según éste esté presente o no. Es una de las funciones más importantes y complejas, ya que deberemos evitar interferir con las autodetecciones de los demás *chipsets* que compongan el servidor.

Dotar al servidor de una función para la detección y selección de los distintos relojes que soporta la tarjeta. Si el generador de reloj es programable, por contraposición a ofrecer una serie discreta de relojes, deberemos incluir el código necesario para programarlo.

Ofrecer una serie de funciones para guardar, restaurar e inicializar la estructura de datos que contiene los valores de los registros.

Proveer al servidor de una función que permita seleccionar la dirección de comienzo de la pantalla dentro del *frame buffer* de la tarjeta, para de este modo poder ofrecer la funcionalidad de *ventana virtual*.

Si el generador de reloj es programable deberemos incluir el código necesario para programarlo

Finalmente, existen dos funciones opcionales en un *driver*: una se encargaría de proveer mecanismos de cambio de consola (*VT switching*) en el caso de que con el mecanismo estándar VGA no funcione de un modo correcto. La otra consistiría en una función que compruebe si los modos que se desean obtener están más allá de las capacidades de la tarjeta.

Después de describir las funciones que debe ofrecer el *driver*, pasaremos a describir en detalle su implementación.

Figura 4. Método utilizado para escribir en los bancos de la tarjeta de vídeo. Los valores de los 8 bits menos significativos de AX (AL) se han movido a los 8 más significativos (AH), introduciéndose el valor del índice en AL.

Mecanismo de escritura en el Puerto E/S

Código de GUSetReadWrite	Datos en AX
VALOR INICIAL	**XXXXXXXX10010001** | Valor nulo | # Banco |
MOV_B (AL , AH) Movemos el banco a los 8 MSB de AX.	**1001000110010001** | # Banco | # Banco |
MOV_B (CONST(8), AL) Cargamos en los 8 LSB de AX el indice del registro Selector de Lectura.	**1001000100001000** | # Banco | S. Lectura |
MOV_B (CONST(0x3DE), EDX) Cargamos en EDX la direccion del registro de Extensión.	**1001000100001000** | # Banco | S. Lectura |
OUT_W Escribimos los 16 bits de AX.	**1001000100001000** | # Banco | S. Lectura |
INC_B (A) Incrementamos AL, seleccionando el registro Selector de Escritura.	**1001000100001001** | # Banco | S. Lectura |
OUT_W Escribimos los 16 bits de AX.	**1001000100001001** | # Banco | S. Lectura |
RET	

El ejemplo que se encuentra en el CDROM de la revista tiene todo el código y ayudará a clarificar los contenidos. Ese código no funciona, de hecho, ni siquiera compila. Este comportamiento es una precaución para evitar que se manipulen accidentalmente registros utilizados por otros dispositivos, generando comportamientos no deseados del servidor de X-Windows. Al fin y al cabo la tarjeta para la que estamos realizando el *driver* es, después de todo, imaginaria.

La tarjeta para la que estamos realizando el driver es imaginaria

• Estructura de datos vgaGURec

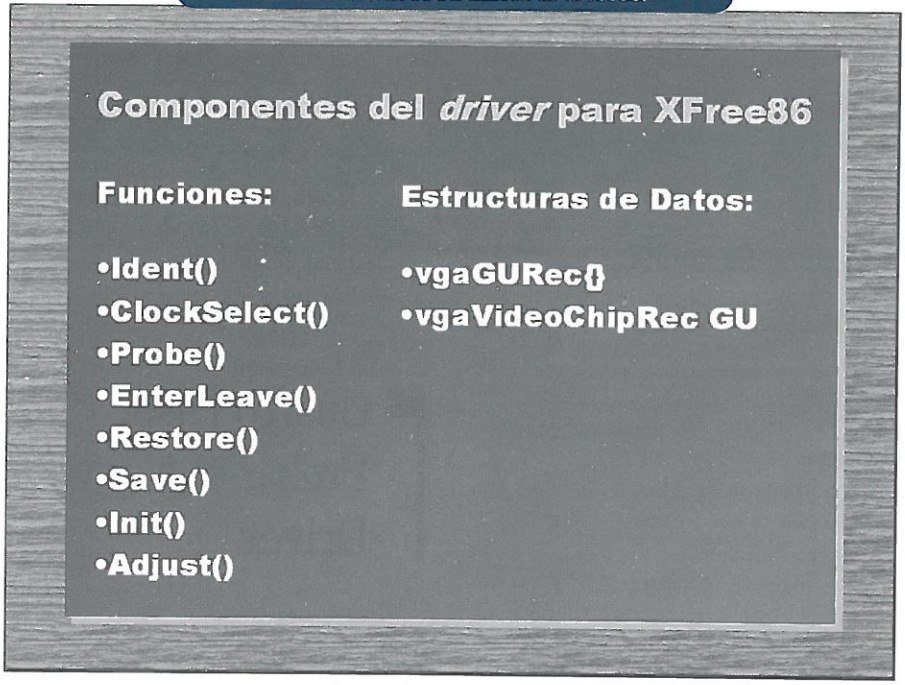
Local al código del *driver*, esta estructura guarda los valores de los registros de la SVGA. El primer elemento es siempre el que se corresponde con los registros VGA genéricos y pertenece al tipo *vgaHWRec*.

• Estructura de datos GU

Esta estructura es de tipo *vgaVideoChipRec* y es la estructura exportada al servidor que informa de las capacidades de la tarjeta. Sus componentes son una serie de punteros a las funciones implementadas en el *driver* y una serie de datos estáticos que enumeran detalles técnicos de la tarjeta. Estos datos son:

- *ChipMapSize*: la cantidad de memoria que debe reservar el servidor. En nuestro caso, y normalmente, es la ventana de 64 Kbytes.
- *ChipSegmentSize*: el tamaño de un banco. En el sistema de bancos utilizado por *Graphics'r'Us* es 64 Kbytes. En sistemas de doble banco de lectura/escritura podría tener diferentes valores.
- *ChipSegmentShift*: determina el número de bits que debemos desplazar hacia la derecha una dirección de vídeo para obtener el número de banco. Su valor es log_2 *(ChipSegmentSize)*.
- *ChipSegmentMask*: máscara utilizada para obtener la dirección dentro de un banco determinado. Su valor se corresponde con *ChipSegmentSize - 1*
- *ChipReadBottom, ChipWriteBottom*: dirección de memoria inferior de los bancos de lectura y escritura respectivamente. para nuestro sistema de bancos será 0xA0000 en ambos casos.
- *ChipReadTop, ChipWriteTop*: dirección de memoria superior de los bancos de lectura y escritura respectivamente. De nuevo será la misma para ambos: 0xAFFFF.
- *ChipUse2Banks*: indica si nuestro *chipset* tiene dos bancos de lectura/escritura. En nuestro caso la respuesta es afirmativa.
- *ChipInterlaceTiming*: puede tener dos valores en función de que el *chipset* necesite que las temporizaciones sean divididas por dos o no en los modos entrelazados. En nuestro caso no lo necesita.
- *ChipRounding*: la pantalla virtual de las tarjetas de vídeo deben ser múltiplos de un número entero que suele ser 8. En algunos casos la documentación de la tarjeta sugiere valores de 16 ó 4, pero suele ser muy extraño.

Figura 5. Componentes imprescindibles de un driver dentro de la colección de servidores de XFree86.



• Función Ident()

Recibe llamadas hasta que devuelve un *NULL*. Se utiliza para informar al *driver* de los *chipsets* que soporta la tarjeta, devolviendo un nombre por cada uno de ellos.

• Función ClockSelect()

Selecciona un reloj u otro dependiendo del argumento. Devuelve *NULL* si el valor está fuera del rango o si el reloj no está disponible. Recibe una serie de parámetros que condicionan su comportamiento para evitar la corrupción del campo de registros de la tarjeta.

• Función Probe()

Debe identificar un *chipset* determinado como único. Para conseguirlo nos ayudaremos de la información contenida en el manual de registros de la tarjeta. Además debe proporcionar otros datos

sobre la tarjeta como puede ser la memoria instalada, el reloj máximo, etc.

• Función EnterLeave()

Se utiliza siempre que se produzca un cambio de consola virtual. Es una función trivial, cuya estructura se puede ver en el código ejemplo.

• Función Restore()

Realiza tres funciones, por este orden:

- Se asegura de que el banco seleccionado es el número 0.
- Restaura los valores de VGA estándar con una llamada a *vgaHWRestore()*.
- Restaura aquellos registros extendidos que sean necesarios para un nuevo modo de vídeo.

• Función Save()

De arquitectura similar a la función anterior, su cometido es salvar los valores de los registros necesarios para mantener un modo de vídeo. Realiza las funciones en el mismo orden:

- Selecciona el banco 0.
- Guarda los valores de los registros VGA con una llamada a *vgaHWSave()*.
- Guarda el contenido de los registros extendidos.

• Función Init()

Se utiliza para inicializar el nuevo modo de vídeo cada vez que éste cambia. Inicializa la estructura *vgaGURec* con los valores necesarios para el nuevo modo. Los nuevos valores que han de tener los registros VGA se obtienen, como siempre, con una llamada. En este caso la función es *vgaHWInit()*.

Toda la información sobre el modo al que el servidor desea conmutar se pasaría en el parámetro llamado *mode*, que es un puntero a una estructura del tipo *DisplayModeRec*.

• Función Adjust()

De nuevo una función muy básica. El ejemplo debería ser autoexplicativo, además de genérico para la mayoría de *chipsets*.

Una vez hayamos realizado todas las funciones aquí descritas, habremos acabado con la tarea de codificación. A partir de aquí solo nos resta compilarlo, probarlo y, en caso necesario, depurarlo.

Cuarto Paso: Probar el Nuevo Driver

Una vez acabada la codificación llegó el momento de probar el código que ya tenemos acabado. Para ello volveremos al directorio raíz del *LinkKit* y prepararemos los *Makefiles* para la compilación con la orden:

./mkmf <ENTER>

Una vez los *Makefiles* estén correctamente creados, y si no hemos obtenido ningún mensaje de error, podremos compilar el nuevo servidor tecleando: **make <ENTER>**

Tras unos minutos de compilación, que serán más o menos dependiendo de la potencia de la CPU y la cantidad de memoria que tengamos, deberíamos obtener un fichero llamado **XF86_SVGA**. Éste es el servidor recién compilado. En algunos sistemas que no tengan los *includes* bien instalados, es posible que el compilador se queje de que no encuentra algún fichero *.h*.

Típicamente el fichero culpable es *stddef.h*. La solución es buscarlos dentro del directorio del *gcc* y realizar un enlace simbólico a ellos dentro del directorio *include/* del *LinkKit*.

Ahora es el momento de probar el nuevo servidor, antes de instalarlo en el sistema. Los resultados de un servidor mal codificado son bastante curiosos y suelen acabar colgando el PC. Por ello es aconsejable hacer un par de *sync* antes de probarlo, así como desmontar todos los sistemas de ficheros no imprescindibles. Finalmente, y tras cruzar los dedos, podemos probarlo tecleando: **./XF86_SVGA <ENTER>**

Si todo funciona, veremos la textura inicial de las X-Windows y un cursor, pero nada más. Podemos volver a modo texto para ver si la traza ha sido correcta y para lanzar aplicaciones. Una vez nos cercioremos de que funciona podemos instalarlo con **make install <ENTER>**.

Conclusiones

A lo largo del artículo hemos aprendido cómo funciona una tarjeta de vídeo por dentro y hemos realizado un servidor básico SVGA para un ejemplo imaginario, capaz de ofrecernos modos de resolución altos en 8 bpp. En el próximo artículo de la serie expandiremos el trabajo aquí iniciado, añadiendo soporte para funciones típicas de los aceleradores actuales: rellenado rápido, cursor *hardware*, *linear framebuffer*, etc. Sólo me queda recordar que el código ejemplo (*gu_driver.c)*no es más que eso, un ejemplo. Utilizarlo tal cual es peligroso, ya que existe la posibilidad de colgar el sistema si no se tiene cuidado.

Contactar con el autor

Jorge Delgado Mendoza es miembro de XFree86 desde 1994, como desarrollador responsable del servidor de Oak Technologies Inc. Es Ingeniero Superior de Telecomunicaciones por la UPM y programador profesional. Puede ser contactado **ernar@dit.upm.es**.

Creación de componentes: trucos y nuevas nociones importantes

Enrique de la Lastra Leralta

Todos sabemos lo que cuesta conseguir que un programa funcione correctamente. Y no sólo eso, lo más difícil es lograr que funcione correctamente *en cualquier caso*. La cuestión es que una vez terminado el grueso del programa, después de mucho sudor y quebraderos de cabeza, no habremos hecho sino empezar, pues lo más difícil comienza en ese preciso instante: conseguir que el programa no falle después de cualquier combinación de acciones por parte del usuario. Si no hemos sido cuidadosos al escribir el código, con un buen estilo de programación, proveyéndolo de comentarios allí donde hicieran falta, la labor de prueba y depuración puede durar indefinidamente y llegar a ser un "auténtico calvario".

Cuando se programan componentes (o de forma más genérica, cuando se programan clases) todo lo anterior se multiplica por un factor, digamos "N" (tendiendo "N" a infinito cuando la experiencia se aproxima a cero). Las razones de por qué esto es cierto son varias:

1. Durante todo el proceso de creación de un componente no podemos "ver" nada hasta que no está terminado (al menos cuando no está acabado el "grueso" del componente). Por tanto, podemos pasarnos varios días programando a ciegas sin saber si estamos siguiendo el camino correcto o incluso si lo que intentamos es acertado y posible.

2. Al programar un componente debemos pensar en el usuario del mismo, es decir, en el programador de aplicaciones (que la mayoría de las veces seremos nosotros mismos, pues el componente lo diseñaremos a nuestra medida). Esto conlleva un pequeño cambio de mentalidad: ya no programamos para un usuario, al que en principio debemos considerar un desconocedor de la informática y al que debemos ofrecer una aplicación que resulte muy fácil de usar; ahora programamos para un programador al que debemos mostrarle la interfaz adecuada y esconderle todo aquello que no queremos que toque.

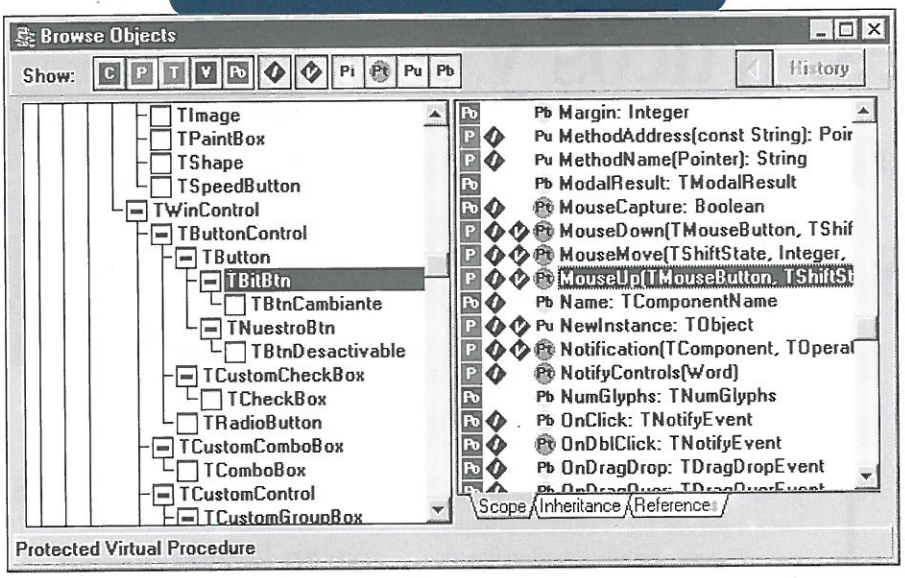
3. La prueba y depuración de un componente conforman un proceso muy complejo debido a que ya no se reduce a "ejecutar y probar". Ahora para probar un componente deberemos añadirlo a una aplicación y probar desde ésta que todo funciona correctamente. Para efectuar estas pruebas tenemos dos opciones:

   3.1. Añadirlo a la paleta de componentes y desde ahí a un proyecto Delphi.
   3.2. Crearlo manualmente en un proyecto Delphi, sin necesidad de añadirlo a la paleta, y compilar el proyecto.

En el segundo caso obtenemos como ventaja un proceso de prueba más

Los seres humanos pasan la vida en una sala oscura donde la mayoría permanece tranquila, inmóvil; pero unos pocos (los sabios) se levantan para investigar entre la oscuridad y... se dan golpes con las paredes. La programación se parece a la sala oscura. Los "sabios" investigan los detalles aprendiendo a base de "golpes". Si lees evitarás alguno.

Figura 1. Clase "TBitBtn" mostrada en el "Browser".



rápido, pero tenemos la limitación de no poder probarlo en tiempo de diseño (que no siempre funcionará igual que en tiempo de ejecución, ya sea porque no es posible o porque lo hayamos programado así). En el primer caso, sin embargo, podremos probarlo en el entorno de desarrollo de Delphi pero deberemos recompilar la librería cada vez que cambiemos el componente, lo cual lógicamente es un proceso que lleva más tiempo.

4. Por último, cabe reseñar el problema que puede ocurrir cuando demos por finalizado, probado y depurado el componente y, sin embargo, contenga un error. Pasado un tiempo, cuando ese error se manifieste, será difícil descubrir si se debe a nuestro componente o a la aplicación que lo utiliza.

Así pues, cada vez que creamos un componente nos enfrentamos a un complejo proceso de programación que tiene como premio final el conseguir una pieza de código muy fácilmente reutilizable. Cuanto más sepamos de las particularidades de Delphi, menos nos costará resolver los problemas que vayan surgiendo.

En este artículo aprenderemos unos cuantos trucos y conceptos importantes aplicables a la programación general en Delphi y otros particulares de la creación de componentes. Sabremos cómo utilizar el *Browser* (hojeador) que incorpora Delphi. Veremos dos formas diferentes de redefinir un evento en un componente. Se explicarán las diferencias entre los componentes visibles y los no visibles y los componentes que pueden coger foco y los que no. Describiremos cómo heredar nuestros componentes de otros de los que no tenemos el código fuente (es decir, de componentes de los que no tenemos el fichero *.PAS y sólo el código objeto, esto es, el fichero *.DCU). En definitiva, trataremos una serie de conceptos que nos pueden resultar muy útiles a la hora de programar.

El browser de Delphi

Para comenzar, vamos a detenernos a conocer una herramienta que incorpora Delphi y que resulta de gran utilidad para el diseñador de componentes. Esta herramienta es el *Browser* (hojeador) de objetos, que permite ver toda la jerarquía de clases de Delphi así como todas las clases que hayamos creado nosotros (incluidos los formularios). Para mostrar el *Browser* se selecciona en el menú "View" la opción "Browser". En la Figura 1 se muestra el *Browser* de Delphi (de la versión 1.0) mostrando el contenido de la clase "TBitBtn".

En la parte izquierda aparece la jerarquía de clases de Delphi, donde resulta claro cuál es la clase padre de una dada y cuál la clase padre de aquélla. En el ejemplo vemos como "TBitBtn" hereda directamente de la clase "TButton", la cual hereda de "TButtonControl", que a su vez hereda de "TWinControl". Esto nos facilita la decisión de por dónde empezar a heredar el componente que queramos crear. A la derecha se muestran las variables, constantes, métodos, eventos y propiedades de la clase seleccionada en la parte izquierda, además de informar de si son heredadas y/o virtuales y de si son privadas, protegidas, públicas o publicadas. Si queremos mostrar sólo las propiedades, pero no los métodos ni eventos ni demás, utilizamos la botonera que se muestra en la

Pulsando estos botones haremos que se muestre en el panel derecho del *Browser* los elementos correspondientes de la clase. Los cinco primeros botones permiten seleccionar respectivamente: Constantes, Procedimientos, Tipos, Variables y Propiedades. Los dos botones siguientes habilitan: el primero, la visualización de las funciones o eventos que la clase seleccionada hereda de la clase padre y el segundo, las funciones que son virtuales. Por último, los cuatro botones de más a la derecha muestran respectivamente las partes privadas, protegidas, públicas y publicadas de la clase. En las fi-

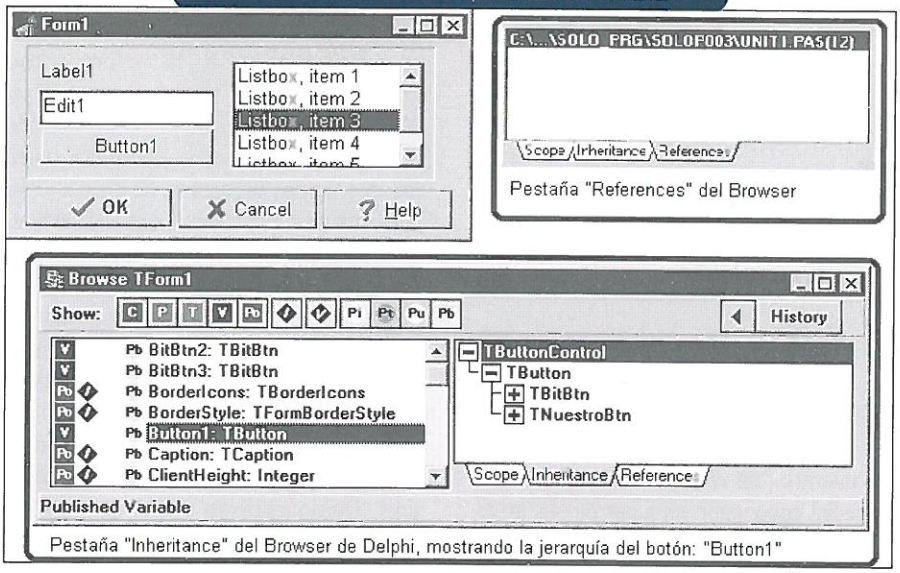
Figura 2. Caja de botones del "Browser" en la que aparecen seleccionadas las propiedades que sean published.

El "Browser" de Delphi nos permite adentrarnos en la estructura de las clases de la Librería de Componentes, así como en la de las clases que creamos nosotros

guras 2 y 3 se muestra un ejemplo para el que sólo aparecen en el *Browser* las variables de la clase seleccionada definidas en la parte "private".

Tenemos, además, otras utilidades disponibles con esta herramienta. En el panel derecho aparecen tres pestañas etiquetadas: "Scope" (que es la que hemos visto), "Inheritance" (herencia) y "References" (referencias). Para entender la utilidad de estas dos últimas creamos un formulario (Form1) como el que podemos ver en esta misma página en la figura 4.

Figura 4. Formulario de ejemplo y contenido de la pestaña "Inheritance" del Browser para dicho formulario.

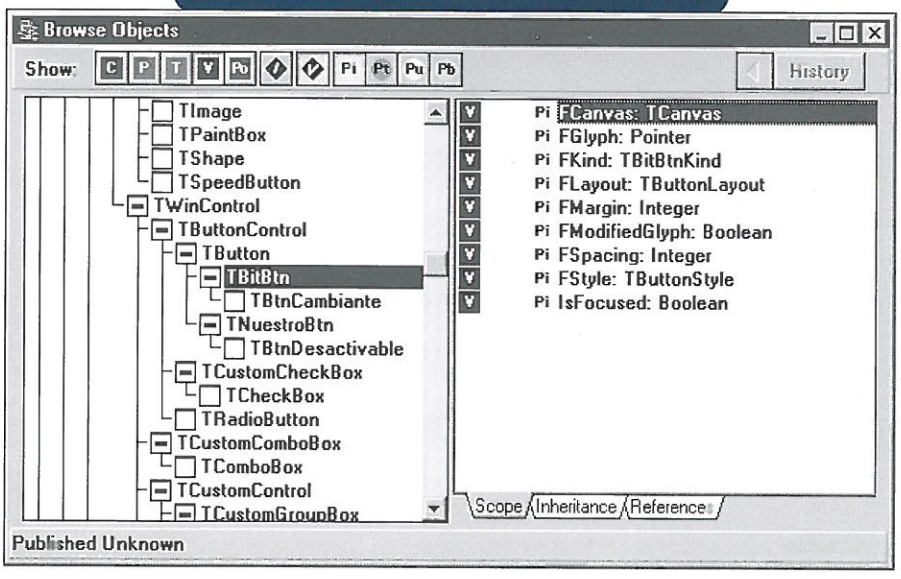


Compilamos el proyecto para que el *Browser* pueda conocer qué clases contiene. Después buscamos la clase del formulario (TForm1) en la estructura de clases que se muestra en el *Browser* y hacemos doble click en la clase "TForm1". Pulsamos la pestaña "Inheritance". Al seleccionar cualquier objeto perteneciente al formulario, aparecerá en el panel de la derecha la jerarquía de clases para la clase a la que pertenezca dicho objeto, como se muestra en la figura 5 para el objeto "Button1". Si pulsamos la pestaña "References" (ver figura 5) aparecerán en el panel derecho los archivos en los que hay objetos de la clase a la que pertenece "Button1" (TButton).

¿Y todo esto para qué sirve? Cuando queremos crear un componente nuevo, encontramos muchas respuestas para esta pregunta:

1. Conocer los métodos y eventos que podemos redefinir (aquéllos que son "virtual").
2. Saber hasta qué orden de la jerarquía está definido un método o evento, para saber desde dónde nos conviene heredar nuestra clase.
3. Publicar (es decir redefinir en la parte "published" o en la parte "public") aquéllas propiedades o eventos que en la clase padre no son accesibles por sus instancias (pues están definidas en la parte "protected"). De esta forma los objetos (instancias) de la nueva clase que creemos sí podrán acceder a esas propiedades o eventos.
4. Conocer cómo está estructurada una clase de la que no tenemos el código fuente (se explicará más adelante).
5. Por último, pero no por ello menos importante: aprender cómo se organiza una jerarquía de clases

Figura 3. Browser mostrando sólo las variables privadas.



compleja, en la que se va ofreciendo funcionalidad desde las clases base y se va ampliando en las hijas. Todo ello realizado por un equipo de profesionales.

Redefinir eventos

Para comenzar, vamos a ver dos formas posibles de redefinir un evento y la manera de utilizar la más fácil siempre y cuando sea posible. Recordamos la definición de evento: "un evento es una propiedad especial cuyo valor, en lugar de ser un tipo, es un método de reacción ante un suceso". Por ejemplo, el evento "OnClick" es del tipo "TNotifyEvent" que se define como:

```
TNotifyEvent = procedure (Sender: TObject) of
    object;
```

Por tanto, la variable que recoge el evento "OnClick" es un puntero a un método del tipo "TNotifyEvent". Pues bien, existen componentes estándar que llevan implementado dentro de la clase alguno de estos métodos y que, además, son "virtual" (lo que permite redefinirlos). Por ejemplo, la clase "TBitBtn" ya tiene un método "Click", virtual, construido dentro de la misma. Así pues, si en el componente que vayamos a crear queremos utilizar el método "OnClick" podremos hacerlo: redefiniendo el evento "OnClick" de la clase padre o redefiniendo el método "Click" del padre (si está presente en dicha clase).

Vemos un ejemplo de los dos casos posibles creando un componente sencillo. Creamos un botón, heredero de "TBitBtn" que cambie el color de su fuente (que por defecto es de color negro) al pulsar el ratón (es decir, en el evento "OnMouseDown") y vuelva a cambiar de color al soltar el ratón (es decir, en el evento "OnMouseUp"). Si observamos la clase "TBitBtn" en el "Browser" vemos que tiene los métodos "MouseDown" y "MouseUp" heredados de "TButton" y que además éstos métodos son "virtual" y "protected" (ver Figura 1).

Llamaremos a la nueva clase "TBtnCambiante". Vamos a redefinir el evento "OnMouseDown" y el método "MouseUp" para ver la diferencia entre los dos modos de actuar. El primer caso es similar a lo visto en el anterior artículo. La única diferencia es que ahora el evento "OnMouseDown" es de la clase "TMouseEvent" y, por tanto, los parámetros del evento y del método que lo trata, han de ser los definidos en el susodicho tipo. Para los que no tienen el artículo anterior, explicamos brevemente el código correspondiente: lo que hacemos es redefinir "OnMouseDown", crear una variable que recoja el método ("FOnMouseDown"), crear nuestro propio método para manejar el evento ("MouseDownInterno") e introducir las líneas necesarias para que las instancias de la clase puedan utilizar dicho evento.

Para el segundo caso, redefinimos el método "MouseUp" (acción que podemos hacer por ser un método "virtual" de la clase padre) añadiendo las siguientes líneas de código en la parte "protected" de la clase (lo hacemos en la parte "protected" porque no queremos que este método sea accesible en los componentes, instancias, de nuestra clase):

```
protected
procedure MouseUp (Button: TMouseButton;
  Shift: TShiftState;
  X, Y: Integer); override;
```

Recordemos que la palabra clave "override" consigue que podamos redefinir un método o evento, sin perder el acceso al mismo. Como podemos observar, el método no lleva como parámetro:

Listado 1 a. Código de un botón que cambia de fuente al ser pulsado y se restaura la antigua fuente al soltarlo.

C:\DELPHI\LIB\BTNCAMB.PAS

```
unit BtnCamb;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls, Buttons;

  type
  TBtnCambiante = class(TBitBtn)
  private
    FOnMouseDown: TMouseEvent;
    procedure MouseDownInterno (Sender: TObject; Button:
              TMouseButton; Shift: TShiftState; X, Y: Integer);
  protected
    procedure MouseUp (Button: TMouseButton;
              Shift: TShiftState; X, Y: Integer); override;
    constructor Create (AOwner: TComponent); override;
  published
    property OnMouseDown: TMouseEvent read FOnMouseDown
                                      write FOnMouseDown;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Standard', [TBtnCambiante]);
end;
```

2: 1 Insert BtnCamb

Listado 1 b. Código de un botón que cambia de fuente al ser pulsado y se restaura la antigua fuente al soltarlo.

C:\DELPHI\LIB\BTNCAMB.PAS

```
constructor TBtnCambiante.Create (AOwner: TComponent);
begin
  inherited Create (AOwner); {Create del padre}
  { Recogemos el evento de la clase padre y lo tratamos }
  inherited OnMouseDown := MouseDownInterno;
  { Definimos nuestro evento, para que no apunte a nada }
  FOnMouseDown := nil;
end;

procedure TBtnCambiante.MouseDownInterno (Sender: TObject; Button:
          TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  { Cambiamos el color de la fuente a color ROJO }
  Font.Color := clRed;
  { Si está asignado, devolvemos el evento }
  if Assigned (FOnMouseDown) then
     FOnMouseDown (Sender, Button, Shift, X, Y);
end;

procedure TBtnCambiante.MouseUp (Button: TMouseButton;
          Shift: TShiftState; X, Y: Integer);
begin
  { Volvemos a cambir el color de la fuente a color NEGRO }
  Font.Color := clBlack;
  { LLamamos al método de la clase padre }
  inherited MouseUp (Button, Shift, X, Y);
end;

end.
```

19: 56 Insert

BtnCamb

"Sender: TObject", ya que como es un método, el "Sender" es el propio objeto que lo utiliza. Además, como no redefinimos el evento sino el método, no hace falta introducir ninguna línea extra en el constructor de la clase.

En el primer caso, para "devolver" el evento de forma que pueda ser manejado por las clases hijas, escribimos dentro de nuestro método interno llamado "MouseDownInterno" :

```
procedure TNuestroBtn.MouseDownInterno
      (Sender: TObject; Button: TMouseButton;
      Shift: TShiftState;
      X, Y: Integer);
begin
 Font.Color := clRed;
 if Assigned (FOnMouseDown)
then
 FOnMouseDown (Sender, Button, Shift, X, Y);
end;
```

En el segundo lo que hay que hacer es llamar al método de la clase padre (MouseUp) que es el que realiza el procesamiento importante:

```
procedure TNuestroBtn.MouseUp (Button:
      TMouseButton;
      Shift: TShiftState;
      X, Y: Integer);
begin
 Font.Color := clBlack;
 inherited MouseUp (Button, Shift, X, Y);
end;
```

La palabra reservada *inherited* provoca una llamada al método "MouseUp" de la clase padre (es decir de "TBitBtn"). El cambio de color de la fuente es inmediato, como se puede comprobar.

El código completo del componente se muestra en el Listado 1. Instalamos el componente, para lo cual: seleccionamos en el menú *Options* la opción "Install Components..." si trabajamos con Delphi 1.0, o en el menú Component la opción "Install Components..." si trabajamos con Delphi 2.0. Pulsamos el botón "Add...", luego el botón "Browse...", seleccionamos el nombre del fichero que contiene nuestro componente y pulsamos *OK*.

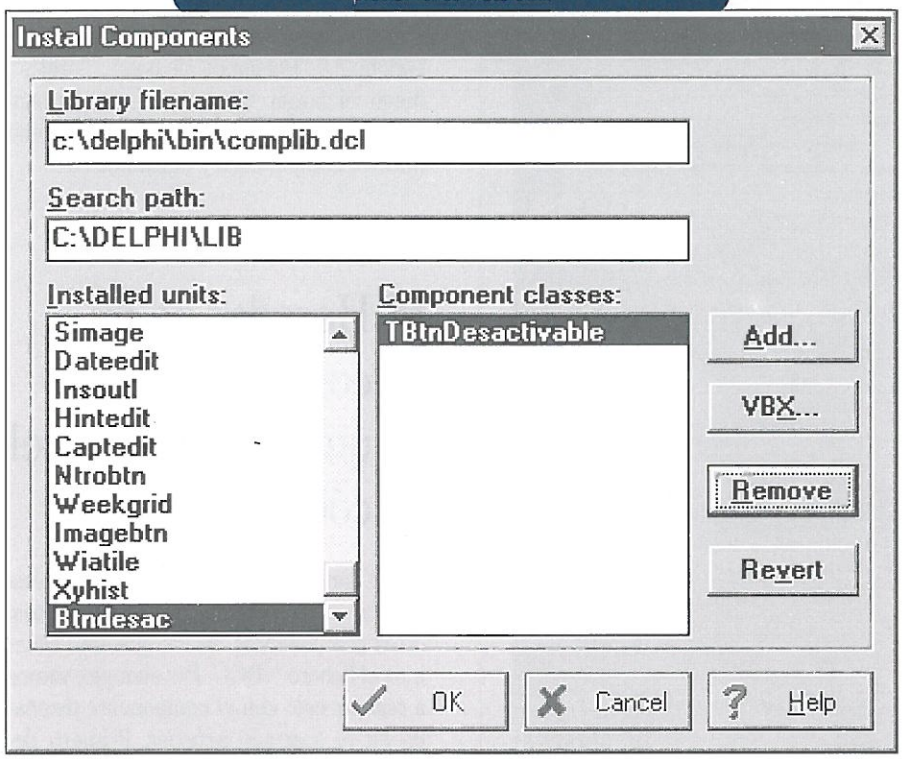
Heredar de un componente del que no tenemos el código

Como ya comentamos, se pueden instalar en la paleta de componentes, otros componentes de los que sólo tenemos el fichero *.DCU. Por ejemplo, vamos a realizar esto con el componente diseñado en el apartado anterior. Primero desinstalamos dicho componente, seleccionando: "Options/Install Components..." en Delphi 1.0, o "Component/Install Components..." en Delphi 2.0. En el cuadro de diálogo que aparece (ver) elegimos la "unit" correspondiente donde pone "Installed units:"; pulsamos el botón "Remove" y luego el botón "OK". La librería de componentes se recompilará, quitando este componente de su código.

Una vez terminada la recompilación, repetimos el proceso pero en lugar de pulsar "Remove" pulsamos "Add..." y luego "Browse...". En el cuadro de diálogo siguiente elegimos en "Mostrar archivos de tipo:" la opción "Unit file (*.DCU)". A partir de aquí el proceso de instalación es idéntico al que efectuamos normalmente.

Cuando el componente esté instalado y sólo entonces, podremos inspeccionar su diseño en el "Browser". Es importante recalcar que sólo podremos ver en el "Browser" aquéllos componentes que estén instalados en la paleta de componentes de Delphi, pues únicamente en ese momento puede conocer el sistema cuál es la estructura interna de la clase.

Figura 5. Cuadro de diálogo para desinstalar el componente previamente instalado.



Si abrimos el "Browser" y elegimos la clase de nuestro componente ("TBtnCambiante") y además no mostramos lo que venga heredado de la clase padre (es decir dejamos sin pulsar el botón "I", "inherited") veremos lo que aparece en la Figura 6. Es decir, sabremos qué propiedades, variables, métodos y eventos se han añadido a dicho componente y todo ello sin necesidad de disponer de su código fuente.

Componentes no visibles

Hasta ahora sólo hemos visto componentes que heredan de la clase "TButton" o de la clase "TBitBtn". Sin embargo, como sabemos, podemos escoger como clase padre de nuestro componente cualquiera dentro de la jerarquía que ofrece Delphi (recordad que por defecto, todas las clases heredan de "TObject"). Esta jerarquía de clases se muestra en la figura 7. No obstante debemos heredar al menos de "TComponent" para tener las propiedades y eventos mínimos que necesita un componente para funcionar en Delphi. En cualquier rama a partir de "TComponent" iremos obteniendo facilidades adicionales.

Mediante el "Browser" podemos conocer cómo ha sido diseñado un componente de cuyo código fuente carecemos, pero sí tenemos el código compilado (*.DCU)

Si elegimos como clase padre a "TComponent", el componente será no visible. Ejemplos de componentes no visibles son "TMenu" o "TTimer". Este último por ejemplo, permite fijar un temporizador y disparar un evento cuando dicho temporizador vence y sin embargo, el componente no es visible en tiempo de ejecución. Sólo resulta visible en tiempo de diseño, lo que nos permite, de forma fácil, dar valores a sus propiedades y métodos de reacción ante sus eventos.

Vamos a diseñar un componente no visible sencillo. Crearemos un componen-

Figura 6. Muestra de las propiedades, métodos y eventos definidos en un componente del que no tenemos el código fuente.

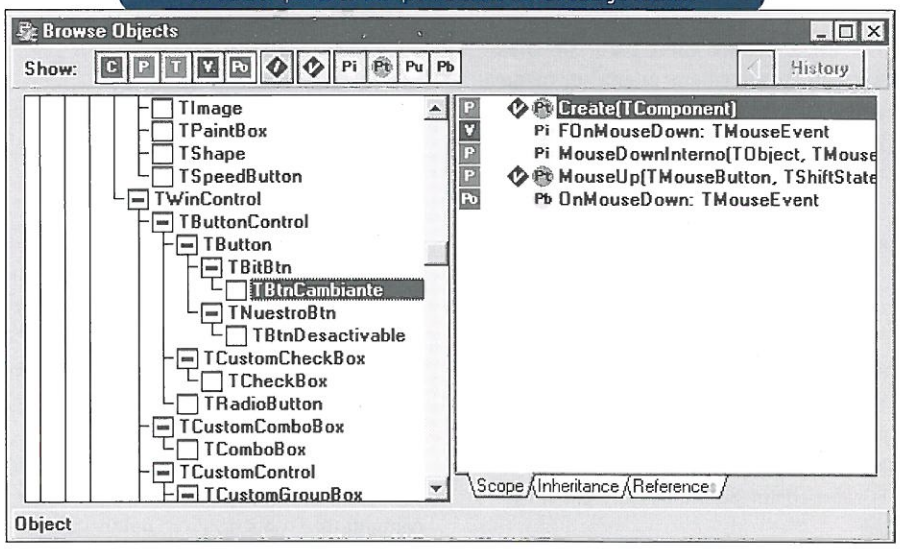
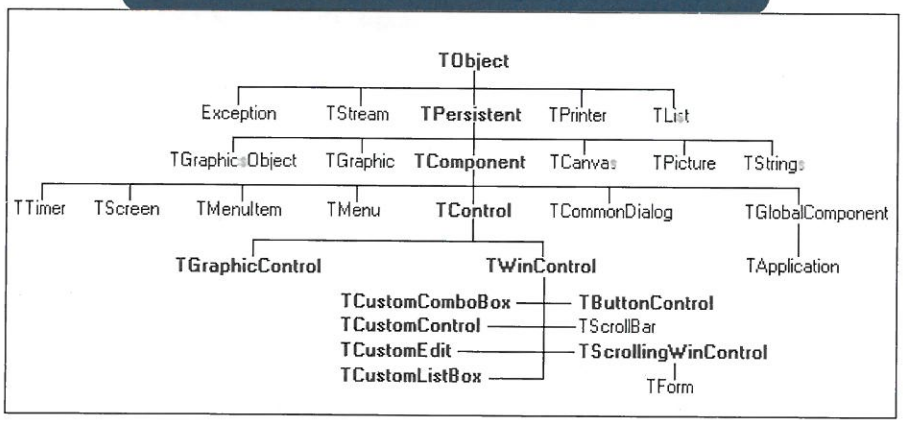


Figura 7. Jerarquía de clases de Delphi.



te que calcule el logaritmo de cualquier número en cualquier base, al que llamaremos: "TCalcLogComponent" y heredaremos directamente de "TComponent". Necesitaremos dos propiedades: la base y el número, que llamaremos "LogBase" y "LogNumero". La base será un número entero mayor que cero, mientras que el número será un valor en coma flotante, que deberá ser también mayor que cero, por lo que habrá que realizar una comprobación de este extremo antes de validarlo (en caso de que cualquiera de los dos sea negativo, no se hará el cálculo). Además definiremos como propiedad la variable que obtenga el resultado (para así observar el funcionamiento en tiempo de diseño).

Declararemos, por tanto, en la parte *private* tres variables: dos serán las que recojan los valores de la base y el número y una obtendrá el resultado:

```
private
FLogBase: Integer;
FLogNumero: single;
FResultado: single;
```

Declararemos, además, en la parte *published*:

```
published
Property LogBase: Integer read FLogBase write
      SetLogBase;
Property LogNumero: single read FLogNumero
      write SetLogNumero;
Property Resultado: single read FResultado write
      FResultado;
```

La propiedad "LogNumero" debe ser de tipo "single" y no "real" o "double" o "extended", pues si no, se producirá un error al intentar editarla en el *Object Inspector*.

Los procedimientos "SetLogBase" y "SetLogNumero" deberán ir en la parte *private*:

```
private
Procedure SetLogBase   (AValue: integer);
Procedure SetLogNumero (AValue: single);
```

En el constructor inicializaremos las variables "FLogBase" y "FlogNumero" a -1, para no realizar la operación hasta que no haya valores válidos:

```
constructor TCalcLogComponent.Create (AOwner:
      Tcomponent);
begin
...
 FLogBase := -1;
 FLogNumero := -1.0;
...
end;
```

Es importante hacer notar que las variables en coma flotante se deben escribir con un "punto" (7.5 por ejemplo) cuando se escriben por programa y sin embargo se deben escribir con una "coma" al editarlas desde el *Object Inspector* (7,5 en el ejemplo).

Ahora sólo hay que introducir el código que se va a ejecutar cuando modifiquemos las dos propiedades

```
Procedure TCalcLogComponent.SetLogBase
      (AValue: integer);
begin
if AValue > 0 then begin
   FLogBase := AValue;
   if FLogNumero > 0.0 then
    FResultado := Ln(FLogNumero) /
      Ln(FLogBase);
 end;
end;

Procedure TCalcLogComponent.SetLogNumero
      (AValue: single);
begin
 if AValue > 0.0 then begin
   FLogNumero := AValue;
   if FLogBase > 0 then
    FResultado:= Ln(FLogNumero) /
      Ln(FLogBase);
 end;
end;
```

En estos dos procedimientos comprobamos que el valor asignado a la propiedad es mayor que cero y sólo en ese caso empleamos la fórmula para calcular el resultado (Fórmula: Logaritmo en base "B" de un número "N" es igual al Logaritmo neperiano (Ln) del número "N" dividido por el "Ln" de "B").

Como la función "Ln" se define en el módulo "System", habrá que añadir esta "unit" a la parte "uses".

El código completo del componente se muestra en el Listado 2. Después de instalar el componente ya sólo tenemos que añadirlo a un formulario, dar valor a las propiedades "LogBase" y "LogNumero" y observar en el *ObjectInspector* cómo cambia la propiedad: "Resultado".

Componentes que pueden coger foco

En general los componentes que utilizamos normalmente pueden coger foco, lo cual significa que podemos acceder a ellos a través del ratón o del teclado.

Componentes que pueden coger foco son, por ejemplo: "TButton", "TEdit", "TForm",.. Sin embargo, hay otros componentes que no pueden recibir foco, como son "TImage" o "TLabel". El aspecto más importante de un componente que puede coger foco es que tiene un *handle* (manejador) a su ventana windows, metido en una propiedad denominada "Handle". Esto significa que Windows conoce el control y permite pasarle dicho *handle* a las funciones de la API de Windows (que lo necesitan para saber con qué ventana están operando).

Los componentes "normales" (aquéllos que pueden coger foco) heredan, por lo menos, de la clase "TWinControl", mientras que el resto (los componentes "gráficos") heredan al menos desde "TControl". Sin embargo, si vamos a diseñar un componente que no vaya a coger foco es mejor heredarlo de "TGraphicControl". La razón es que si heredamos de "TGraphicControl" tendremos la propiedad "Canvas" disponible para "pintar" el componente y además podremos manejar los mensajes Windows: WM_PAINT, por lo que bastará hacer

Listado 2 b. Componente no visible que calcula el logaritmo en cualquier base.

C:\DELPHI\LIB\CALCLOG.PAS

```
constructor TCalcLogComponent.Create (AOwner: TComponent);
begin
  inherited Create (AOwner); {Create del padre}
  FLogBase := -1;
  FLogNumero := -1.0;
end;

procedure TCalcLogComponent.SetLogBase (AValue: integer);
begin
  if AValue > 0 then begin
    FLogBase := AValue;
    if FLogNumero > 0.0 then
      FResultado := Ln(FLogNumero) / Ln(FLogBase);
  end;
end;

procedure TCalcLogComponent.SetLogNumero (AValue: single);
begin
  if AValue > 0.0 then begin
    FLogNumero := AValue;
    if FLogBase > 0 then
      FResultado:= Ln(FLogNumero) / Ln(FLogBase);
  end;
end;

end.
```

33: 1 Modified Insert

Calclog

"override" del método "Paint". Todo esto no está definido en "TControl". Para elegir de qué clase vamos a derivar nuestro componente, tenemos que considerar:

1. Si el componente no necesita coger foco, es mejor que sea un componente gráfico (es decir heredero de "TGraphicControl"), así ahorramos recursos del sistema al no necesitar *handles* a sus ventanas.

2. En caso contrario, heredaremos el componente de "TWinControl" o de cualquier clase hija de ésta. Así consumiremos más recursos Windows.

En el próximo artículo sobre creación de componentes, diseñaremos un componente gráfico, heredándolo directamente de "TGraphicControl".

Listado 2 a. Componente no visible que calcula el logaritmo en cualquier base.

C:\DELPHI\LIB\CALCLOG.PAS

```
unit Calclog;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs;

type
  TCalcLogComponent = class(TComponent)
  private
    FLogBase: integer;
    FLogNumero: single;
    FResultado: single;
    procedure SetLogBase   (AValue: integer);
    procedure SetLogNumero (AValue: single);
  public
    constructor Create (AOwner: TComponent); override;
  published
    property LogBase: integer read FLogBase write SetLogBase;
    property LogNumero: single read FLogNumero write SetLogNumero;
    property Resultado: single read FResultado write FResultado;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Standard', [TCalcLogComponent]);
end;
```

2: 1 Modified Insert

Calclog

Contactar con el autor

Si desea contactar con el autor envíe un e-mail a la dirección de la revista o a miluen@redestb.es e indique como *subject* : "DELPHI SOLOP" (en mayúsculas y con un espacio).

Interfaces, serialización y, de postre, threads

Oscar Prieto Blanco

Alicia no habría tenido esos problemas, si se hubiera encontrado en *JavaLand*. En este mundo ficticio, tenemos el poder de recorrer diferentes caminos simultáneamente que nos conducirán, de forma concurrente, hasta el final de la tarea, del algoritmo, que estemos intentando implementar.

¿Hechicería, Vudú, magia negra, complicados y peligrosos ritos iniciáticos? Por supuesto que no, simplemente nos desprenderemos de nuestra fina carcasa mortal y de la misma manera que las Moiras griegas hilaban, devanaban y cortaban el hilo de la vida de los hombres, nosotros aprenderemos a crear y manipular las distintas hebras de ejecución de los programas (*threads*) que nosotros creamos.

Pero un procesador solamente tiene dos manos, así que ¿cómo consigue realizar varias tareas al mismo tiempo? Mediante un truco de prestidigitación, ya que es lo suficientemente rápido como para permitirse el lujo de saltar (o conmutar) entre las diferentes tareas a ejecutar. De tal manera que nos engaña a los pobrecitos humanos con el mismo truco que vienen empleando los sucesores de los hermanos Lumiere, desde principios de siglo.

Antes de comenzar con los *threads*, debemos conocer ciertas cosillas acerca de la utilización de las interfaces, así que sin más dilación...

Interfaces y herencia múltiple

En el artículo del mes anterior tropezamos con las clases abstractas. Su utilización da soporte a la idea de un concepto general, por ejemplo *Shape*, de la que sólo las variantes más concretas (clases derivadas), como *Point* o *Circle*, podrán ser instanciadas realmente. Es decir, una clase abstracta es una clase que se puede utilizar, únicamente, como clase base de otra. Para declarar una clase abstracta contamos con el modificador *abstract*. Mediante una clase abstracta intentamos fijar un conjunto mínimo de métodos (el comportamiento) y de atributos, que nos permitan modelar un cierto concepto, que será refinado y especializado adecuadamente mediante el mecanismo de la herencia. Como consecuencia, la implementación de la mayoría de los métodos de una clase abstracta podría no tener significado.

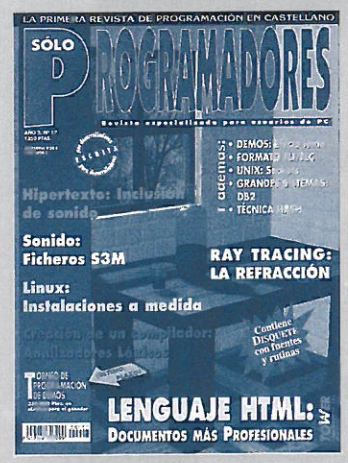
Java proporciona un mecanismo para resolver este dilema: los métodos abstractos. Estos métodos se encuentran incompletos, sólo cuentan con la declaración y no poseen cuerpo de definición. La *keyword abstract* debe preceder a la declaración de estos métodos. Una clase que contenga uno de estos métodos es una clase abstracta y debe ser calificada como tal, lo inverso no es necesariamente cierto, una clase abstracta no está obligada a tener algún método abstracto.

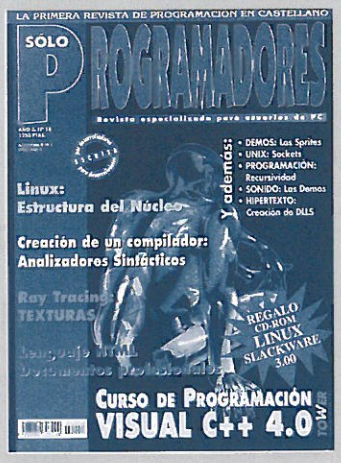
> "¿Me podrías indicar a dónde tengo que ir?", preguntó Alicia. "Eso depende de a dónde quieras llegar", contestó el Gato. "A mí no me importa a dónde...". "En ese caso, da igual hacia donde vayas ... siempre que llegue a alguna parte". "¡Oh! Siempre llegarás a alguna parte, si caminas lo bastante".
>
> Alicia en el País de las Maravillas.
>
> Lewis Carroll

números atrasados

PROGRAMADORES

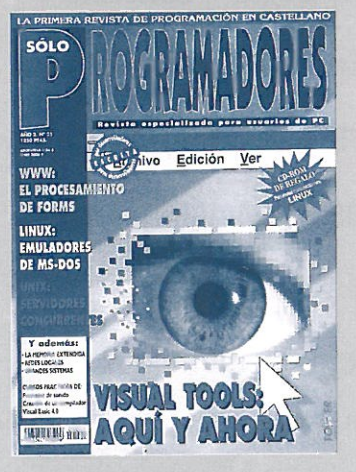
completa ya tu colección

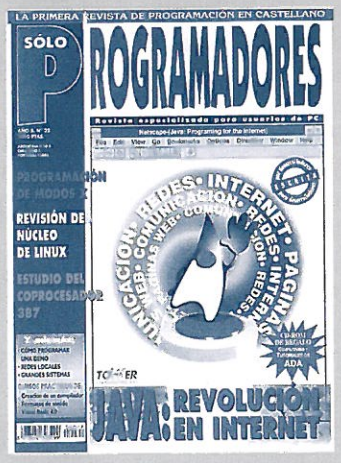

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
Sonido: Ficheros S3M
RAY TRACING: LA REFRACCIÓN
Linux: Instalaciones a medida
LENGUAJE HTML: DOCUMENTOS MÁS PROFESIONALES


LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
Y además:
• DEMOS: Las Sprites
• UNIX: Sockets
• PROGRAMACIÓN: Recursividad
• SONIDO: Las Demos
• HIPERTEXTO: Creación de DLLS
Linux: Estructura del Núcleo
Creación de un compilador: Analizadores Sintácticos
REGALO CD-ROM LINUX SLACKWARE 3.00
CURSO DE PROGRAMACIÓN VISUAL C++ 4.0

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
REGALO CD-ROM LOTUS GALLERY
Lenguaje HTML
Instalación de servidores WWW
LINUX: Device Drivers
Y además:
• DEMOS: Blur Motion
• SONIDO: Filtros digitales en Tiempo real
• RAY TRACING
CURSO DE PROGRAMACIÓN VISUAL BASIC 4.0

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
UNIX Aplicaciones distribuidas
VISUAL C++ 4.0 La librería de clases MFC
WEB
CGI: EL INTERFAZ DEL FUTURO


LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
WWW: EL PROCESAMIENTO DE FORMS
LINUX: EMULADORES DE MS-DOS
Y además:
VISUAL TOOLS: AQUÍ Y AHORA


LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
REVISIÓN DE NÚCLEO DE LINUX
ESTUDIO DEL COPROCESADOR 387
JAVA: REVOLUCIÓN EN INTERNET

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
EL PODER DE LOS 32 BITS
ESTUDIO DE LOS APPLETS DE JAVA
LINUX: CÓMO CREAR DISCOS DE EMERGENCIA A MEDIDA
WATCOM C++

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
ESTUDIO DEL FORMATO WAV
LAS VENTANAS LLEGAN AL WEB
DEPURACIÓN DE CÓDIGO EN VISUAL C++
PROGRAMACIÓN DEL PUERTO SERIE

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
LINUX: PRESENTACIÓN DE TEXTO EN PANTALLA
WEB: DESCRIPCIÓN AVANZADA DE INTERFAZ CGI
TÉCNICAS DE COMPRESIÓN
ALGORITMO HUFFMAN

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
LINUX: EL GESTOR DE VENTANAS X-WINDOWS
WEB: LENGUAJE HTML
EUSKAL PARTY 96
EL FESTIVAL DE LA DEMOSCENE

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
ANALIZAMOS LOS PROCESADORES DEL MERCADO
EXCEPCIONES EN DELPHI
VISUAL J++
LA APUESTA DE MICROSOFT

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
ANALIZAMOS EL ESTADO DE LA INTELIGENCIA ARTIFICIAL
VISUAL C++ PROGRAMACIÓN DEL GDI
ESPECIAL HERRAMIENTAS RAD
DESARROLLO RÁPIDO DE APLICACIONES CON OPTIMIZACIÓN

suscríbete

a Sólo Programadores y consigue un magnífico descuento

suscripción normal

ahorro **20%**

12 revistas (1 año) por sólo...

9.350 ptas.

suscripción estudiantes
(carreras técnicas)

ahorro **40%**

12 revistas (1 año) por sólo...

7.050 ptas.

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
BORLAND INTRABUILDER PROFESIONAL 1.0
COMUNICACIONES EN WINDOWS

Veamos el clásico ejemplo de las formas geométricas:

```
abstract class Shape { // Nuestra clase base
  public double area() { return 0.0; }
  public double volume() { return 0.0; }
  // Método a definir en clases derivadas
  public abstract String getName();
}

// Un punto es una figura geométrica.
class Point extends Shape {
  protected double x, y; // Coordenadas
  // constructor
  public Point(double a, double b)
  { setPoint(a, b);}
  // Métodos principales
  public void setPoint(double a, double b)
  { x = a; y = b; }
  public double getX() { return x; }
  public double getY() { return y; }
  public String toString()
  { return "[" + x + ", " + y + "]"; }
  public String getName() { return "Point"; }
}

// Un círculo es un punto (centro) y algo más.
public class Circle extends Point {
  protected double rad; //Radio
  // Constructores
  public Circle()
  { super(0, 0); setRadius(0); }
  public Circle(double r, double a, double b)
  {super(a, b); setRadius(r); }
  // Métodos típicos
  public void setRadius( double r )
  { rad = ( r >= 0 ? r : 0 ); }
  public double getRadius() { return rad; }
  public double area(){ return 3.14*rad*rad;}
  public String toString()
  { return "Centro = " + super.toString() +
          "; Radio = " + rad; }
  public String getName() { return "Circle"; }
}

// A partir de un círculo obtenemos un cilindro
class Cylinder extends Circle {
  protected double height; // Altura
  public Cylinder( double h, double r,
          double a, double b)
  { super( r, a, b ); setHeight( h ); }

  // Métodos
  public void setHeight( double h )
  { height = ( h >= 0 ? h : 0 ); }
  public double getHeight() { return height; }
  public double area(){
    return 2 * super.area() +
        2 * 3.14159 * radius * height; }
  public double volume()
  { return super.area()*height;}
  public String toString()
  { return super.toString()+"; Altura =
      "+height;}
  public String getName() { return "Cylinder"; }
}
```

Java se diseñó con la idea de que fuera un lenguaje "sencillo" y por tanto se le denegó la capacidad de la herencia múltiple, tal como es conocida en otros lenguajes como el C++. En este lenguaje se añade cierta complejidad sintáctica cuando realizamos herencia múltiple de varias clases, las cuales comparten una clase base común (debemos declarar dicha base como virtual y tener bastante cuidado con los constructores), o también cuando las clases base tienen miembros de nombre similar (entonces hay que utilizar especificadores de acceso).

Java simula la herencia múltiple mediante el uso de interfaces. Una *interfaz* sublima el concepto de clase abstracta hasta su más alto grado. Una *interfaz* podrá verse simplemente como una forma, es como un molde, sólo nos permite declarar nombres de métodos, listas de argumentos, tipos de retorno y adicionalmente miembros datos (los cuales podrán ser únicamente tipos básicos y serán tomados como constantes en tiempo de compilación, es decir *static* y *final*). El usuario que desee implementar una *interfaz* deberá definir cada una de las declaraciones de los métodos ofrecidos por dicha *interfaz*. En el listado1 podemos observar la sintaxis y principales propiedades de las interfaces.

Una *interfaz* puede heredar de varias sin ningún problema. Sin embargo una clase sólo puede heredar de una clase base, pero puede implementar varias interfaces. También tenemos la posibilidad de definir una *interfaz* vacía, como es el caso de *Serialize*, que está provisto por el JDK 1.1.1. y que nos permite serializar un objeto.

Una *interfaz* vacía se suele utilizar como un *flag*, esto es, para marcar a una clase con una propiedad determinada. Veamos un ejemplo de cómo puede aplicarse esto último:

```
// PipaTest.java
interface FumadorPipa{}

class P{
  public static void prt(String s)
  { System.out.println(s); }
}

class Persona{
  public String name;
  public Persona(String s){ name = s;}
}

class Fumador extends Persona
        implements FumadorPipa{
  public Fumador(String s){ super(s);}
}

class PipaClub{
  private static void Check(FumadorPipa fp){}
  static void Welcome1(Persona per){
      try{
    PipaClub.Check((FumadorPipa)per);
       P.prt("Bienvenido: "+per.name);
      }catch(Exception e)
{ P.prt("Hasta luego," +per.name);}
  }
  // Otra forma de chequear al fumador:
  static void Welcome2(Persona per){
      if(per instanceof FumadorPipa)
       P.prt("Bienvenido: "+per.name);
   else
       P.prt("Hasta luego," +per.name);
  }
}

public class PipaTest{
  public static void main(String args[]){
      Persona p = new Persona("Juan");
      Fumador f = new Fumador("Oscar");
      PipaClub.Welcome2(f);
      PipaClub.Welcome1(p);
  }
}
```

Si observamos el listado 1 nos tropezamos con el concepto de serialización de objetos, así que pasemos a estudiarlo un poco más detenidamente.

Listado 1

```
// InterfTest.java  - Aprendiendo
// sobre interfaces y serialización.
import java.io.*;

interface Pack
{      byte[] pack();  }
interface UnPack
{      void unpack(byte raw[]);}
// Un interface puede heredar de varios
interface PackMachine extends Pack, UnPack {}

class Human implements Serializable{
  StringBuffer sbName;
       // Este atributo no se serializará:
       transient Byte age;
       public Human()   // ctor. por defecto
  {    sbName = new StringBuffer();
       age = new Byte((byte)0);
  }
       public Human(String sName, byte age){
       sbName = new StringBuffer(sName);
       this.age = new Byte(age);
  }
}
// La herencia múltiple soportada por Java:
class Person extends Human
      implements PackMachine, Serializable
{
public int DNInum;
public Person(){} // ctor. por defecto
public Person(String s, int dni, byte age){
       super(s, age);
       DNInum = dni;
}
  // Nos sirve de ayuda en la función pack()
  private byte p(int t, int n)
  {    return (byte)((t>>n) & 0xff); }
  // Empaqueta los atributos en un array de bytes
  public byte [] pack(){
       byte ret[] = new byte[5+sbName.length()];
       ret[0]=p(DNInum,24); ret[1]=p(DNInum,16);
       ret[2]=p(DNInum,8);  ret[3]=p(DNInum, 0);
       ret[4] = age.byteValue();
       for(int j=0;j<sbName.length();j++)
         ret[5+j] = (byte)sbName.charAt(j);
     return ret;  // Retornamos un array
  }
  // Ayudante de la función unpack()
  private int u(byte b, int n)
  {    return (b & 0xff) << n;  }
  // Desempaqueta el array de bytes:
  public void unpack(byte raw[]) {
     DNInum = u(raw[0], 24)|u(raw[1], 16)
              |u(raw[2], 8)|u(raw[3], 0);
     age = new Byte(raw[4]);
     sbName = new StringBuffer();
      for(int i=5;i<raw.length;i++)
        sbName.append((char)raw[i]);
  }
  // Par imprimir información por pantalla:
  public String toString(){
      if(age == null){
        return "Person: " + sbName +
             ", " + DNInum + ", "+ "XX";
      }
      return "Person: " + sbName +
           ", " + DNInum + ", "+ age;
  }
 }  // Fin de la clase Person
// Una clase para hacer pruebas:
public class InterfTest{
  public static void main(String args[])
  {
   Person p1 = new Person("Oscar",
                          9778996,(byte)26);
   System.out.println(p1);
   byte prueb[] = p1.pack();
   Person p2 = new Person(); p2.unpack(prueb);
   System.out.println(p2);
   try{
     ObjectOutputStream out =
       new ObjectOutputStream(
         new FileOutputStream("person.out"));
      // Escritura de un obj. serializable:
      out.writeObject("Una persona persistente"+
                      "y coqueta:\n");
      out.writeObject(p1);
      out.close(); // importante
      ObjectInputStream in =
        new ObjectInputStream(
          new FileInputStream("person.out"));
      // Lectura de un obj. serializable:
      String s = (String)in.readObject() ;
      Person pSerz = (Person)in.readObject();
      System.out.println(s + pSerz);
   }catch(Exception e)
   {  e.printStackTrace();  }
  }
}// Fin de la clase InterfTest
```

Serialización de objetos

Esta característica ha sido añadida a la versión 1.1 y, gracias a ella, tenemos la posibilidad de tomar cualquier objeto que implemente la *interfaz Serializable* y transformarlo en una secuencia de bytes, que podrán ser escritos en un *stream* de salida y posteriormente restaurados y leídos en un *stream* de entrada. ¿Qué ventajas podemos obtener con esta nueva forma de actuar? Pues siempre hemos contado con la posibilidad de trabajar con *streams* de entrada y salida.

- La primera ventaja es de orden conceptual. De este modo se adopta un enfoque que está orientado hacia los objetos, en un apartado que, hasta esta versión, tenía que ser codificado de manera artesanal (con todos los riesgos que ello implica; el azar, afirmó Darwin, afecta a la selección natural, y no siempre obtendremos una mutación en el código que resulte positiva).

  A partir de este momento, en la tarea de serializar un objeto nos es transparente la estructura interna del mismo. Esto significa que, gracias a esta nueva técnica, no sólo podremos salvar una imagen de nuestro objeto, sino que también todos los *handles* que estén contenidos en nuestro objeto son rastreados y salvados. Y, más todavía, realiza el mismo proceso con cada uno de los *handles* que estuviesen contenidos en esos objetos, etc. Claro está, siempre que todos esos objetos implementen la *interfaz Serializable;* en el caso contrario, será arrojada una excepción.

  A toda esta maraña de objetos se la suele conocer con el nombre de grafo de objetos o bien *web* de objetos y, hasta la llegada de la serialización, existían ocasiones en que era bastante difícil seguir la pista de todos los objetos cuando queríamos salvarlos de una manera segura.

- Mayor comodidad cuando se esté trabajando en una red. Por ejemplo, podríamos crear un objeto en una máquina Windows, serializarlo (esto es, transformarlo en un flujo de bits) y enviarlo a través de la red a una máquina Unix, donde será correctamente reconstruido (la ciencia ficción ya ha llegado hasta nosotros, querido Spock), sin habernos preocupado de sus detalles de arquitectura, tales como la diferente representación interna que tienen los datos en las distintas máquinas (*little endian, big endian).*

- Nos permite implementar una propiedad conocida como persistencia. Gracias a la persistencia, entendemos que el tiempo de vida de un objeto no está determinado por la circunstancia de que un programa se esté ejecutando o no. Un objeto (una instancia singular de una clase, para ser más exactos) conservará sus propiedades entre todas las diferentes invocaciones que tenga el programa en cuestión.

  Para lograr esto, simplemente tomamos un objeto serializable y lo escribimos en disco. Posteriormente, cuando el programa sea llamado de nuevo, podremos restaurar el estado del objeto con una simple lectura del disco.

En el listado 1 podemos observar la sencillez con la que es puede serializar un objeto.

Creamos algún tipo de *OutputStream*, en este caso *FileOutputStream* (para que podamos trabajar con ficheros), aunque bien podía haber sido *ByteArrayOutputSream*, con el que utilizaríamos un *buffer* en memoria. Seguidamente lo envolvemos dentro de un objeto *ObjectOutputStream* y a partir de este momento ya podemos utilizar funciones tales como *writeObject()*, que nos permite serializarlos objetos (así como los *arrays* y las *Strings*) o funciones típicas como *writeByte(), writeInt(),* etc., para los tipos básicos.

Para que sea posible ejecutar el proceso inverso, tendremos que envolver alguna subclase de *InputStream* dentro de un objeto *ObjectInputStream* y a continuación llamarlo a funciones tipo *readObject()* o similares. No hay que olvidar, por lo demás, que en cualquier caso debemos seguir un orden análogo al tomado al escribir los objetos y recordar aplicar siempre el *casting* oportuno.

Debemos tener en cuenta una serie de detalles cuando vayamos a implementar el mecanismo de serialización en nuestros programas. Mediante la *keyword transient* será posible fijar los distintos componentes de nuestras clases que no deseemos serializar. En el listado 1, como yo soy un poco coqueto, lo aplico al *Byte* age. Esto significa que, a la hora de reconstruir el objeto, el compilador lo inicializará por defecto. Así pues, en este caso, tomará el valor *null*. Situación que es aprovechada a la hora de codificar la función *toString():*

```
public String toString(){
  if(age == null)
      return "Person: " + sbName +
            ", " + DNInum + ", "+ "XX";
  return "Person: " + sbName +
          ", " + DNInum + ", "+ age;
}
```

Las subclases de una clase serializable también lo son (ésta es una consecuencia básica de las propiedades que tienen las interfaces). Por otro lado, subclases de objetos que en principio no son serializables pueden ser serializadas. En este caso, la clase que no es serializable tendrá que contar con un constructor sin argumentos accesible (*public* o *protected*) que permita inicializar sus campos y que será llamado por el compilador en el momento en que deba reconstruir el objeto en cuestión. En estas ocasiones, es responsabilidad nuestra el escribir y leer los datos de esa clase base en los *streams*, bien accediendo directamente a los datos o, en caso de que esto último no sea posible, mediante alguna función tipo *getXXX*, que nos devuelva información sobre ellos. Un ejemplo de todo esto se puede ver en el siguiente listado:

```
import java.io.*;

class Inutil{
        private int k; public Inutil(){}
        public int getVal(){return k;}
        public void setVal(int i){ k = i;}
}

class Estupid extends Inutil
implements Serializable{
   private int ar[];
   private static int r()
   {   return (int)(Math.random()*15);}
   public Estupid()
   { setVal(r()); ar = new int[r()]; }
   public void SetData()
   { for(int i=0;i<ar.length;i++) ar[i] = r();}
   public void PrintData(){
    System.out.println("Idiota num: "+getVal());
    for(int i=0;i<ar.length;i++)
        System.out.print(ar[i]+" ");
    System.out.println(); // Importante
   }
}

// Nuestro campo de pruebas
public class Serz{
   public static void main(String args[]){
    Estupid s = new Estupid();
    s.SetData(); s.PrintData();
    try{
        // Escritura:
        ByteArrayOutputStream bout =
            new ByteArrayOutputStream();
        ObjectOutputStream out =
            new ObjectOutputStream(bout);
        out.writeObject(s);
        out.writeInt(s.getVal());
        out.flush();
            //Importante
            // Lectura:
        ObjectInputStream in =
            new ObjectInputStream(
              new ByteArrayInputStream(
                bout.toByteArray() ));
        Estupid s1 = new Estupid();
        s1 = (Estupid)in.readObject();
        s1.setVal(in.readInt());
        s1.PrintData();
      }

catch(Exception e)
        { e.printStackTrace(); }
   }
}
```

Como comentario final de este apartado, es muy importante reseñar que existe todavía otra *interfaz*, que es conocida como *Externalizable y* que extiende a la ya de por sí completa *Serializable* añadiéndole dos métodos distintos: *writeExternal()* y *readExternal(),* los cuales son automáticamente llamados para nuestros objetos, durante los procesos de serialización y deserialización, permitiéndonos conformar el proceso a nuestra medida. Un pequeño ejemplo de ello:

```
class Simple implements Externalizable{
  String s; int k;
  public Simple(){} // De vital importancia
  public Simple(String st, int i)
  {
  s = st;
  k = i;
  }
  public void writeExternal(ObjectOutput out)
   throws IOException{
    out.writeObject((s+" ole, ole"));
    out.writeInt(++k);
  }

  public void readExternal(ObjectInput in)
   throws IOException, ClassNotFoundException
  { s=(String)in.readObject();
  k=in.readInt();}
  public String toString(){return s + k; }
}
```

Esta clase en estos momentos podría ser serializada con los hoy en día ya clásicos métodos *writeObject() y readObject(),* pertenecientes a *ObjectOutputStream* y *ObjectInputStream*, respectivamente. Un único detalle debe ser tenido en cuenta a este respecto. La clase que se encargue de la implementación de *Externalizable* deberá contar, siempre, con un constructor que no tenga argumentos públicos. El porqué de esta imprescindible norma se encuentra en el hecho de que, cuando se tiene entre manos algún objeto *Externalizable,* antes de que se produzca la llamada a *readExternal,* se ejecutan las inicializaciones en el punto de la definición de sus miembros de datos y, seguidamente, se llama al constructor sin argumentos de dicho objeto (motivo por el cual obligatoriamente tiene que ser posible realizar el acceso a él).

Threads (hebras de ejecución)

En primer lugar, tendremos que empezar por aclarar un poco del vocabulario que vamos a emplear a continuación. Es posible definir un proceso como una instancia de un programa en ejecución. Un *thread* es un flujo, un camino de ejecución dentro de un determinado proceso. Todo proceso, por lo demás, cuenta, como mínimo, con un *thread.* Cada aplicación en Java comienza con un único *thread* en funcionamiento, el cual tiene que ejecutar el método *main* de la clase que ha sido inicialmente especificada en la JVM. Este *thread* podrá crear nuevos *threads.*

Cada uno de los *threads* de un proceso cuenta, por su parte, con su propio *program counter* (registro en el que se almacena la dirección de la siguiente instrucción que tenga que ser ejecutada) y su propio *stack* (en el cual almacena sus variables locales y las direcciones de retorno de procedimientos). En otras palabras, cada *thread* posee su propio contexto de ejecución. Un atributo muy importante de un *thread* es su prioridad. Cuanto mayor sea la prioridad, mayor será la preferencia a la hora de conseguir el acceso a los recursos que posee el procesador.

En Java contamos con dos aproximaciones para trabajar con *threads.* Implementar la *interfaz Runnable* (que cuenta únicamente con un método llamado *run()*), o bien derivar de la clase predefinida *Thread*, provista por este lenguaje. Veamos un ejemplo bastante sencillo de introducción:

```
// InfernalGodess.java
public class InfernalGodess{
  public static void main(String args[])
  {
    MoiraThread th1, th2, th3, th4;
    // Creación de unas infernales diosas
    th1 = new MoiraThread("Cloto");
    th2 = new MoiraThread("Laquesis");
    th3 = new MoiraThread("Atropos");
```

Listado 2

```
// RandomChars.java -> Múltiples threads y
// manejo de eventos con el AWT del jdk 1.1.1

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class RandomChars extends Applet
                implements Runnable
{
  String alphabet;
  TextField outp1, outp2, outp3;
  Button button1, button2, button3;

  // Nuestros threads:
  Thread thread1, thread2, thread3;
  boolean suspend1, suspend2, suspend3;

  // Cuando el applet es primeramente creado,
  //para realizar las inicializaciones oportunas
  public void init(){
    alphabet = new
        String("ABCDEFGHIJKLMNOPQRS-
TUVWXYZ");
    outp1 = new TextField(10);
    outp1.setEditable(false);
    outp2 = new TextField(10);
    outp2.setEditable(false);
    outp3 = new TextField(10);
    outp3.setEditable(false);
    button1 = new Button("Suspend/Resume
1");
    button2 = new Button("Suspend/Resume
2");
    button3 = new Button("Suspend/Resume
3");

    // Añadimos el componente al applet.
    add(outp1);
    // Ligamos a la fuente del evento
    // ActionEvent
    //(el botón) con un "receptor" (Listener)
    button1.addActionListener(new SusRes1());
    add(button1);
    add(outp2);
    button2.addActionListener(new SusRes2());
    add(button2);
    add(outp3 );
    button3.addActionListener(new SusRes3());
    add(button3);
  }

  // Llamado cada vez que el applet aparece en
  //el Web Browser, para permitir que el applet
  //comience sus operaciones
  // normales.Llamado también después de init.
  public void start(){
    // Threads en estado Nuevo_Thread
    thread1 = new Thread( this, "Thread 1");
    thread2 = new Thread( this, "Thread 2");
    thread3 = new Thread( this, "Thread 3");
// Threads en estado Ejecutable_Thread
    thread1.start();
    thread2.start();
    thread3.start();
  }
  // Llamado cada vez que el applet desaparece
  //de la vista del Web Browser, para permitir
  //que el applet detenga sus operaciones
  public void stop(){
    // Threads en estado Muerto_Thread
    thread1.stop();
    thread2.stop();
    thread3.stop();
  }

  // Todo el tomate en la ejecución de un
  // Thread
  public void run()
  {
    int location;
    char disp;
    String execThread;

    while (true) { // Bucle infinito
      // Thread en estado Parado_Thread
      try
      {
      Thread.sleep((int)(Math.random()*5000));
      }catch ( InterruptedException e )
            {  e.printStackTrace(); }

      location = (int)(Math.random()*26);
      disp = alphabet.charAt(location);
  // Con la llamada a currentThread obtengo
  //el thread que se está ejecuntando actualmente
  // execThread =
Thread.currentThread().getName();

      if(execThread.equals("Thread 1"))
        outp1.setText("Thread 1: " + disp);
      else if(execThread.equals("Thread 2"))
        outp2.setText("Thread 2: " + disp);
      else if (execThread.equals("Thread 3"))
        outp3.setText("Thread 3: " + disp);
    }
    // Fin del bucle infinito while
  }

  // Clases anidadas, una nueva caracteristica
  //del JDK 1.1 como ayuda para un nuevo
  // tratamiento
  //del manejo de eventos en el AWT.

class SusRes1 implements ActionListener {
  // Operación a realizar cuando un Listener
  //recibe un evento disparado por un botón.
public void actionPerformed(ActionEvent e){
    if(suspend1)
    {
      thread1.resume(); suspend1 = false;
    }else
    {
      thread1.suspend();
      outp1.setText("Hibernado");
      suspend1 = true;
          }
  }
 } // Fin de SusRes1

class SusRes2 implements ActionListener {
  public void actionPerformed(ActionEvent e){
    if(suspend2)
    {
      thread2.resume(); suspend2 = false;
    }else
    {
      thread2.suspend();
      outp2.setText("Hibernado");
      suspend2 = true;
    }
  }
 } // Fin de SusRes2

class SusRes3 implements ActionListener {
  public void actionPerformed(ActionEvent e){
    if(suspend3)
    {
      thread3.resume(); suspend3 = false;
    }else
    {
      thread3.suspend();
      outp3.setText("Hibernado");
      suspend3 = true;
    }
  }
 } // Fin de clase SusRes3
 static class WL extends WindowAdapter {
  public void windowClosing(WindowEvent e)
{  System.exit(0);
  }
 } // Fin de clase WL

 // Para la ejecución del applet como una
 //aplicación independiente.
  public static void main(String args[]) {
    RandomChars app = new RandomChars();
    Frame aFrame =
        newFrame("Threads&Chars");
    aFrame.addWindowListener(new WL());
    aFrame.add(app, BorderLayout.CENTER);
    aFrame.setSize(300,200);
    app.init();
    app.start();
    aFrame.setVisible(true);
  }
} // Fin de la clase RandomChars
```

```
    // Ojo, es una impostora, es una Parca
    th4 = new MoiraThread("Nona");
    // Arrancamos la ejecución de los threads
      th1.ifMoiraStart();
      th2.ifMoiraStart();
      th3.ifMoiraStart();
      th4.ifMoiraStart();
  }
}

class MoiraThread extends Thread {
  int sleepTime;
  public MoiraThread(String id)
  {
    super( id );
    //El nombre a la clase base.
    // Dormido entre 0 y 5 segundos
    sleepTime = (int)(Math.random()*5000);
    System.out.println("La diosa: "+getName()+
              "; duerme: "+
              sleepTime );
  }

  // ¿Es una Moira?
  public void ifMoiraStart()
  {
      String id = getName();
      if(id.equals("Cloto")|
        id.equals("Laquesis")|
        id.equals("Atropos") )
      {start();
       return; }
      System.out.println("La impostora: "+
          getName() + "; muere.");
      stop();
      // Matamos al thread
  }

  //Donde el thread ejecuta sus tareas
  public void run()
  {
    // Primero lo mantenemos dormido
    try {
      sleep(sleepTime);
    }
    catch (InterruptedException e){
      System.err.println("Exception: "+
                  e.toString() );
    }
      int j = (int)(Math.random()*300);
    System.out.println(getName() +
        " hila, corta y devana la vida de "
          +j+ " hombres");
  }
}
```

Estados de un thread

Un *thread* perfectamente puede ser visto como un objeto con vida propia. Se puede decir que su ciclo de vida es, en cierta manera, el de una máquina de estados finita. En primer lugar, es creado mediante una new; a este estado lo llamaremos Nuevo_Thread (NT). Aquí, el sistema todavía no ha destinado ningún tipo de recurso para él. En este estado, una sencilla llamada a *stop()* lo matará definitivamente (estado Muerto_Thread -MT-), sin ninguna posibilidad de vuelta atrás, y una sencilla llamada a *start()* servirá como catalítico para que comience una serie de reacciones dentro del programa en el que se encuentre.

En primer lugar, se reservarán los recursos del sistema que sean necesarios para que el *thread* pueda ejecutarse. Acto seguido, el *thread* será incorporado a la lista de tareas disponibles para ser ajecutadas que es mantenida por el sistema. Por último, será llamado el método *run()*, que delimita el trozo de código importante en el cual se está produciendo el trabajo, la tarea para la cual haya sido diseñado el *thread* que nos ocupa. Todas las variables que estén declaradas dentro de este método, junto con las direcciones de retorno de las funciones que en él son llamadas, constituyen lo que antes fue apuntado como contexto de ejecución. A este estado lo vamos a designar como Ejecutable_Thread (ET). Ejecutable y no en ejecución, puesto que comparte el procesador con otros procesos y con distintos *threads*, y evidentemente no pueden ser ejecutados todos simultáneamente. El orden en que un conjunto de *threads* es atendido por la CPU es indeterminado, a menos que ajustemos sus prioridades por medio del método *setPriority()*.

Desde el estado Ejecutable_Thread, tenemos dos posibilidades: dirigirnos al estado Muerto_Thread, o bien al estado llamado Parado_Thread (PT). Tendremos la posibilidad de llegar al estado [MT] en el momento en que vayamos a ejecutar el método *stop()* o en caso de que finalice la ejecución del método *run()*. La mayoría de las ocasiones, dentro del método *run()* existirá alguna especie de bucle infinito, lo que implicará que sólo podremos finalizar la ejecución al hacer alguna llamada a *stop()* de manera externa a esta función.

Al estado [PT], podemos llegar de varias maneras distintas. De momento sólo vamos a comentar un par de ellas. Es importante recordar que en este estado no se están consumiendo los recursos del sistema.

Mediante el método *sleep()* dormimos al *thread* una cantidad especificada de milisegundos, retrasándole el tiempo de su ejecución. Cumplido el lapso de tiempo que haya sido especificado, entonces automáticamente retornará al estado [ET]. A no ser que lo matemos, en dicho caso despertará para morir (triste final).

Gracias al método *suspend()* podemos mantener al *thread* en un estado de hibernación del que sólamente podrá salir con una llamada al método *resume()*.

Como ejemplo para comprender toda esta maraña de estados diversos, además del uso de *threads*, vía la *interfaz Runnable* podemos echar una miradita al Listado 2. En él hemos incluido una serie de conceptos nuevos, que todavía no habían aparecido en esta serie y que paso a comentar a continuación.

Es el primer código en esta serie en el que aparece un *applet*. El lector puede consultar el primer número de la serie Cuadernos de programación, entregado en la revista número 34, donde se comentan aspectos básicos de la programación de applets. Aunque me imagino que a estas alturas todo el mundo habrá realizado sus pinitos en páginas Web.

Mediante la adición de un método *main()* a un *applet* conseguimos que se pueda ejecutar como una aplicación independiente:

```
public static void main(String args[])
 {
   // Creamos una instancia del applet:
   RandomChars app = new RandomChars();
   // Un Frame implementa una ventana marco
   //principal de aplicación, que contiene
   //un título y un borde. Adicionalmente
   //puede contener una barra de menús.
   Frame aFrame = new
      Frame("Threads&Chars");
   // Manejamos el evto. De cierre de aplic.
      aFrame.addWindowListener(new WL());
   // La clase Frame deriva de Window que a
   //su vez deriva de Container, luego la
   //podemos añadir componentes.
   aFrame.add(app);
   // Establecemos su tamaño inicial.
   aFrame.setSize(300,200);
   // En un Browser, estas dos funciones son
   //llamadas automáticamente, en una aplic.
   //debemos hacerlo nosotros.
   app.init();
   app.start();
   // Esta función heredada de la clase base
   //Component, lo muestra en pantalla.
   aFrame.setVisible(true);
 }
```

El nuevo modelo de eventos

En el listado 2 nos encontramos con el nuevo modelo de tratamiento de eventos que ha sido introducido en el AWT del JDK 1.1. En este modelo se elimina totalmente la codificación que está basada en sentencias if en cascada, por otra considerada más elegante, mucho más orientada a los objetos. A partir de ahora, contamos con dos tipos de objetos: unos que son fuente de eventos y otros que son receptores de eventos. En esta nueva aproximación, un componente determinado dispara (inicia) un evento. Cada uno de los eventos que tengamos se va a encontrar representado por una clase determinada. Por ejemplo, gracias a la clase *Button* tienes la posibilidad de disparar todos los siguientes eventos: *ActionEvent, FocusEvent, KeyEvent, MouseEvent* y *ComponentEvent*. En efecto, dentro del listado 2 decidimos prestar especial atención al "disparo" del evento *ActionEvent* por parte de los botones. Todos los componentes han sido cambiados para que sean capaces de contener los métodos *addXXXListener()* y *removeXXXListener()*, gracias a los cuales nos es posible ligar y desligar un componente con un receptor de eventos. En el ejemplo mediante la siguiente llamada:

```
Button1.addActionListener(new SusRes1);
```

Realizamos la conexión entre una fuente y un receptor de eventos. La clase pasada como parámetro a esta función deberá implementar la *interfaz XXXListener*, en este caso *ActionListener*. Esta *interfaz* cuenta con una única función: *actionPerformed(ActionEvent)* que, en el caso de un botón, será llamada, como es lógico, sólo en el momento en que sea presionado.

En el listado 2 aparece otra característica novedosa. En la definición de las clases que implementan los *interfaces XXXListener* nos encontramos con que su se encuentra anidada dentro de la clase conocida como *RandomChars*. Esta técnica mejora considerablemente la estructuración del código y vemos que las clases anidadas pueden acceder a todas las variables de sus clases padre y, más aún, toda clase anidada contiene un *handle* a su clase padre (propiedad que, por lo demás, no se encuentra mostrada dentro del Listado 2).

Para finalizar tenemos que comentar un detalle bastante importante. En el manejo del evento *WindowEvent* por parte del componente *Frame* vemos que no se verifican las reglas que hemos visto anteriormente. Esto se debe a que tendríamos que haber escrito WL *implements WindowListener*. Cuando se accede a la documentación del *JDK*, se puede observar que la *interfaz WindowListener* consta de siete funciones, lo que puede ser considerado un considerable engorro si solamente queremos manejar una de ellas, en este caso *windowClosing*. De esta manera tendríamos que definir las otras seis funciones para que realizaran algún tipo de tarea por defecto.

Para resolver este problema, Java nos suministra las clases *Listener Adapters*, que contienen implementaciones por defecto para cada una de las funciones de la *interfaz*. Así que, en estos casos, sólo tenemos que heredar de uno de estos adaptadores y sobrepasar la función que deseemos controlar (en este ejemplo sería *windowClosing()*). La definición de la clase anidada contiene otro pequeño detalle: es estática. De esta manera puede ser creada sin que sea neceasario contar con un objeto de la clase padre. Todo esto será estudiado más a fondo en un próximo artículo.

Conclusión

En este artículo hemos avanzado bastante a través de la tupida selva de APIs que se va formando en torno al lenguaje de programción Java. Comenzamos estudiando las clases abstractas y las distintas *interfaces* como técnicas de reutilización de código. Continuamos con el concepto de serialización que ha sido introducido en la última versión del lenguaje y que le añade muchísimas posibilidades. Para finalizar, hemos arañado un poco la superficie del tratamiento del *multithreading* por parte del lenguaje, lo que nos ha servido como excusa para introducirnos en el nuevo modelo de manejo de eventos del JDK 1.1.

Contactar con el autor

Para cualquier duda, comentario, sugerencia o crítica se anima a los lectores a que se pongan en contacto con el autor de estas páginas a la dirección de E-Mail: myoprieto@redestb.es

RAD Magic 7

Marta Huertas

Las herramientas de desarrollo cada día tienen más importancia, se trata de programar lo mínimo posible, aunque nos tememos que la programación siga siendo inevitable. Estudiemos una RAD que lleva tiempo en el mercado, Magic 7, ¿tiene magia?

Qué es Magic

Magic es un sistema para desarrollar aplicaciones de forma rápida, lo que se suele llamar una RAD (*Rapid Application Development*), que pretende facilitar a los desarrolladores de aplicaciones su trabajo. Como RAD de segunda generación permite el desarrollo de aplicaciones cliente/servidor mediante un entorno de desarrollo conducido por tablas, es decir, sin necesitar código procedural, un editor de texto o un script propietario. Esta forma de generar las aplicaciones de desarrollo permite reducir de forma significativa el tiempo requerido para la codificación, compilación, depuración y recodificación de aplicaciones.

Se trata de eliminar la pérdida de tiempo en la repetición de determinados pasos que se suceden en el llamado ciclo de desarrollo. Así, Magic se concibe como una máquina que ejecuta la lógica de las aplicaciones que se almacenan en su "repositorio", palabreja habitual entre los que utilizan herramientas de desarrollo y que se entiende como una especie de almacén. En el caso que nos ocupa, en este almacén

Bajo un entorno muy sencillo de una única ventana que sirve tanto para desarrollo como para explotación, nos encontramos con un producto de una gran potencia para desarrollar aplicaciones. Sólo tenemos que definirla y una vez creada aparecerán los iconos que nos permiten ir añadiendo elementos en nuestra aplicación. Lo primero que podríamos hacer es definir los tipos, nada más fácil con nuestra herramienta. A base de ventanas seleccionamos los tipos genéricos según los distintos entornos donde nuestra aplicación vaya a ejecutarse. La herramienta permite que el desarrollo se realice de forma concurrente, es decir, que desde distintos puestos de trabajo se acceda a la misma aplicación, esto posibilita, por ejemplo que una vez definido un tipo, el resto de desarrolladores accedan a él seleccionándolo y, si alguno de ellos lo modifica, los demás lo sabrán y el cambio se realizará automáticamente en todas las aplicaciones, programas y bases de datos donde el tipo se utilice.

Una vez definido el tipo, se definen los ficheros, de forma general, ya que si conocemos el sistema específico donde ejecutaremos la aplicación, podremos concretar más adelante los tipos en una lista interminable de ellos. Es importante indicar que todo lo que generamos son tablas, no hay ni una sóla línea de código.

Y así podríamos seguir hasta definir una aplicación completa que puede funcionar en muchos entornos, accediendo a diferentes bases de datos, sin conocer nada sobre ellos. Todo lo que tenemos que hacer es aprender a utilizar esta herramienta cuyo tiempo de aprendizaje estimado es de un mes (para empezar).

se guardarían los elementos que componen una aplicación: tipos de datos, lógica del programa, reglas de producción y presentación de la misma.

Magic es un entorno visual y se basa en una metodología de desarrollo visual, a través de ventanas vamos seleccionando aquellos elementos que necesitamos para definir una aplicación. Las plataformas que soporta son las siguientes: Intel Windows NT, DEC VAX Open VMS, SCO UNIX, IBM AS/400, IBM AIX, SuonOS, SUN Solaris, HP UX, DG UX, Olivetti UNIX SVR4, Siemens SINIX, CTOS, Fujitsu UNIX, NEC UNIX, TICOM UNIX, DG Aviion, DEC Alpha Windows NT, DEC Alpha Open VMS, como plataformas servidores y Windows 3.x y Windows 95/NT como plataformas cliente.

Las bases de datos para las que está preparada son: Oracle, Sybase, Btrieve, C-ISAM, Oracle Rdb, Microsoft SQL Server, DB/2 6000, DB400 para AS/400, Informix, MF COBOL, RM COBOL, ODBC, RMS, xBase. Como se puede observar, para todos los gustos y necesidades.

Características técnicas

En el paradigma de programación de Magic subyacen dos elementos:

- El motor Magic: que proporciona la fuente para alcanzar la productividad y la potencia de Magic, con todas las funcionalidades necesarias para permitir a los desarrolladores la creación de aplicaciones complejas, incluyendo las reglas de producción, la lógica del programa y la presentación. El trabajo de los desarrolladores se limita a cargar y descargar varios de los elementos que Magic les proporciona utilizando una serie de tablas y cajas de diálogo. Los valores como datos, definiciones de pantalla o menús, se convierten en acciones mediante la acción del motor de Magic. Este motor lee y ejecuta el fichero de control de la aplicación en un bucle de ejecución cíclico mediante los ficheros de la aplicación, los registros y los campos. Existe un fichero de control, que es el que determina la lógica del programa.

- Programación automatizada post-4GL: mediante funciones basadas en este tipo de programación se elimina la necesidad de realizar tareas repetitivas, con el consiguiente decremento en el tiempo utilizado en el desarrollo. El desarrollador de la aplicación especifica los nombres, operaciones y parámetros en tablas que automáticamente se actualizan si se produce algún cambio.

En el paradigma de programación de Magic se encuentran la programación automatizada post-4GL y el motor de Magic

Como la mayoría de las RAD, MAGIC hace uso del llamado repositorio o diccionario. En el caso de esta RAD, en forma de un fichero de control que puede ser actualizado según las necesidades, con la descripción de las aplicaciones, datos, programas, formularios, etc.

Las tablas en las que se basa la herramienta también son incluidas en el fichero de control, siendo ésta la forma en la que los cambios que tengan que ver con algún elemento incluido en las tablas, sean conocidos por el resto de los elementos que intervienen en la aplicación, actualizándose de esta forma las bases de datos yel resto de los elementos de la aplicación de forma automática.

Los diccionarios

Los diccionarios de MAGIC los diferentes objetos de la aplicación, que podemos definir mediante el entorno de ventanas:

- Diccionario de datos: la definición de los datos se realiza a través de los datos que proporciona este diccionario. Los desarrolladores definen los datos, ficheros, campos, claves, etc., mediante las tablas que contiene el diccionario, las cuales están centralizadas en el fichero de control de Magic. Las bases de datos pueden ser accedidas de forma simultánea por todos los desarrolladores, ya que se permite el trabajo concurrente sobre una misma aplicación.

- Diccionario de tipos: aquí se encuentran todos los tipos posibles. Un objeto tipo es una definición abstracta de un contenedor de datos que será heredado por otros objetos del mismo diccionario. Además de las características normales de los tipos, se permite definir información relativa a la apariencia que tendrá en pantalla el tipo cuando sea mostrado, las reglas de validación y ventanas de ayuda.

- Diccionario de ficheros: mediante los datos almacenados aquí, se especifican los nombres de las tablas y su localización lógica, junto con detalles de las columnas de las tablas relativas a su almacenamiento en el gestor de la base de datos.

El elemento más importante: las tablas

En Magic todo se realiza mediante tablas, éstas representan las acciones que normalmente se indican mediante código. No tenemos que preocuparnos

por el código ya que mediante tablas se controla la funcionalidad de la mayoría de las aplicaciones que desarrollaremos. De esta forma evitamos uno de los pasos del desarrollo software que consume más tiempo: la programación.

Las tablas se estructuran en cuatro categorías: tablas de ejecución, tablas de campos virtuales, tablas de expresiones y tablas de formulario. Las tablas de ejecución contienen las operaciones que describen la lógica del programa; existirá una tabla por cada uno de los estados por los que pasa el motor de Magic. Las entradas de la tabla definen un identificador de la operación y un conjunto de punteros a otras tablas. En Magic un programa está formado por un conjunto de tareas que se dividen en procedimientos, controlados por el desarrollador, y otros elementos no-procedimentales, que controla el motor de Magic. Cada tarea tiene un modo específico que puede ser modificación, creación, borrado y búsqueda el cual informa al motor de qué tarea debe ser procesada en cada momento. Mediante las tablas de campos virtuales se definen las variables temporales que ser necesitarán en la ejecución de cada tarea. Las tablas de expresiones se utilizan para la definición y evaluación de las operaciones aritméticas, matemáticas, booleanas y de cadena y para proporcionar el acceso a librerías con funciones más completas. Las tablas de formulario se utilizan en el diseño por parte de los desarrolladores de los formularios, para ello, utilizan el editor de formularios que proporciona Magic.

Conclusión

Los problemas que el desarrollo tradicional de software tiene son muchos: los costes de producción sobrepasan habitualmente los inicialmente estimados, la mayoría de los proyectos no cumplen la planificación de tiempo ni de presupuesto y esto constituye un grave problema para programadores y, a veces, para las empresas, tanto desde el punto de vista del cliente como de la empresa de software.

Magic es una solución para estos problemas, incrementa el rendimiento del ciclo de desarrollo tradicional basaso en un análisis, un diseño y un ciclo repetitivo de codificación, compilación y prueba. Con esta RAD, pasamos del análisis a un único paso de diseño del prototipo, prueba y modificaciones, que, una vez dados los últimos retoques, no necesita un mantenimiento costoso en tiempo y dinero.

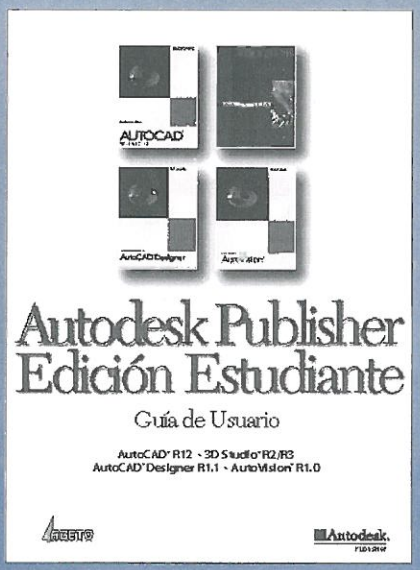
La herramienta es distribuida en España por Askin. Para obtener más información sobre ella:(91) 661 83 04, o bien a través del responsable de producto Miguel Angel Díez.

AUTODESK PUBLISHER EDICIÓN ESTUDIANTE

¿QUÉ PROGRAMAS INCLUYE?

- AutoCAD R12
- 3D Studio R2/R3
- AutoCAD Designer R1.1
- Autovision R1.0.

Y además:
Más de 700 páginas de lecciones de Autodesk con guías de referencia y documentación software on-line



Formato 21x29,7

TODO LO QUE NECESITA PARA EL DISEÑO Y VISUALIZACIÓN PROFESIONAL EN 2D Y 3D LO HALLARÁ EN ESTE PAQUETE DE PROGRAMAS.

EDICIÓN ESPECIAL PARA ESTUDIANTES
19.500 PTAS. (I.V.A. INCLUIDO)

DE VENTA EN LIBRERÍAS Y DISTRIBUIDORES AUTODESK

SISTEMAS ABIERTOS

Servidores de la gama HP 9000

Marta Huertas

Recientemente Hewlett-Packard nos ha presentado un nuevo servidor que pretende revolucionar el tratamiento de los procesos de datos dentro de la empresa. Analizar este servidor es el objeto de este artículo.

Habitualmente los centros de proceso de datos sufren presiones desde el exterior de las organizaciones donde se encuentran. Los componentes de las mismas demandan día a día manejar cantidades mayores de información, en su control está la fuerza, y las empresas solicitan un mejor servicio, quieren rentabilizar sus inversiones en nuevas tecnologías para realizar la gestión de sus datos. Los clientes se vuelven más exigentes y necesitan más requerimientos en el control de su información. En el ámbito de las tecnologías la competencia es brutal, continuamente se investigan nuevas herramientas y aparecen en el mercado productos renovados que dicen ser los mejores.

El cambio de siglo ha traído también un problema añadido: el efecto 2000 supondrá la modificación parcial y a veces el rediseño total de las aplicaciones y con ellas de las bases de datos. Internet es el remate para llegar a la necesidad del cambio, una oleada de nuevos enfoques ha llegado hasta nosotros y nos pide a gritos una evolución. Desde Hewlett-Packard se tuvieron en cuenta todos estos condicionantes a la hora de sacar al mercado nuevos productos.

La empresa extendida

Una empresa que consigue dar soluciones a todo lo que anteriormente hemos expuesto, obteniendo ventajas en los negocios mediante el control de la información, es lo que desde HP se llama empresa extendida. Como primera forma de aproximación de las empresas hacia la consecución de esos objetivos, se indican los siguientes:

- Construir o mejorar un *Web Site*.
- Encaminar los sistemas internos de la empresa hacia la Web.
- Permitir el acceso a través de la Web a sus principales aplicaciones.
- Hacer una reestructuración de su negocio.

La empresa extendida es aquella capaz de dominar el proceso de la información de tal forma que le permita obtener beneficios en los negocios

Como producto ideal para cumplir las expectativas en el crecimiento y acercamiento de la empresa a esa concepción de la empresa extendida, HP propone sus servidores empresariales HP 9000, que en la actualidad son el principal soporte

MASTERS DE INFORMÁTICA

ENTORNO UNIX

LA OBRA DE REFERENCIA MÁS COMPLETA Y ACTUAL

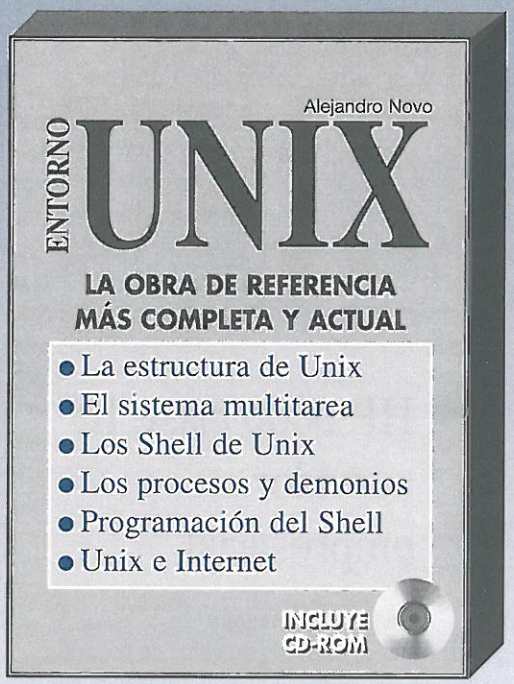
EL ENTORNO UNIX es:

- Uno de los sistemas operativos más extendidos.
- Es el candidato ideal para los servidores.

En esta obra podrá conocer en profundidad todas las particularidades de este sistema, contrastando la mayoría de sus variantes.

Tendrá asegurado un aprendizaje rápido y efectivo gracias al CD-ROM que acompaña el libro, el cual incluye:

- Todos los ejemplos que aparezcan en el texto,
- Programas de autor,
- Utilidades para Unix,
- Una versión completa de Linux.



PRECIO: 4.995 Ptas.

LIBRO + CD-ROM

ABETO EDITORIAL

c/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (Madrid)
Tel.: (91) 661 42 11* - Fax: (91) 661 43 86

para Oracle, Informix y SAP R3. Para corroborar la idoneidad de esta opción HP indica que sus servidores RISC/UNIX de Gestión fueron los más vendidos durante 1996 frente a otras opciones como Sun, IBM, Tandem, BULL o DEC que se repartieron el resto del mercado.

Para continuar con este éxito de ventas, desde HP no dejan de investigar en la mejora de sus servidores. Recientemente han anunciado el nuevo Superservidor Corporativo HP 9000 Clase V y Empresarial Clase K.

HP 9000 Clase V: servidor corporativo

Como producto muy potente, los nuevos servidores reúnen las características requeridas para cubrir las necesidades de una empresa en expansión, en cuanto a su centro de datos. Las características técnicas del servidor son las siguientes:

sistema de multiproceso simétrico de hasta 32 vías con la nueva tecnología *cross-bar HP Hyperplane*. Esta tecnología está compuesta por una placa de alta capacidad de ejecución, puede soportar hasta un máximo de 16 transacciones simultáneas entre la memoria y el procesador en paralelo, y permite el proceso de los datos a una velocidad de hasta 1,9 GB/s en entrada/salida.

basado en el procesador PA-8200 RISC de 64 bits de HP.

opera sobre una nueva versión del sistema operativo HP-UX 11.00. Esta versión de Unix ofrece una implementación completa de 64 bits que incrementa la capacidad de escalabilidad a medida que se hace necesario. Este SO es el límite en la evolución de los SO de HP, al desarrollar las APIs de 64 bits, con tipos de datos, capacidad elevada de memoria (32 GB de RAM), memoria compartida y compatibilidad binaria de las aplicaciones, unido a las funciones internet de que ya disponía HP-UX 10.20.

En los nuevos servidores se puede desarrollar en C++ o en Java, desde un entorno de sistemas abiertos

Con estas características, estos servidores tienen la potencia necesaria para conseguir todo lo que puede hacer un mainframe pero sin serlo, con lo cual se permite el desarrollo de aplicaciones internet e intranet, el desarrollo en C++ o Java, la integración de NT y Windows, junto con los sistemas propietarios, el comercio electrónico o la arquitectura cliente/servidor, dentro de un entorno de sistemas abiertos, donde como procesador se utiliza el RISC con tecnología CMOS (como ya mencionamos), el sistema operativo es UNIX, las comunicaciones utilizan los protocolos TCP-IP, la programación se puede desarrollar en lenguajes de Cuarta Generación, o en C++ y Java, etc.

HP 9000 clase K: servidor empresarial

Como complemento de la clase V, HP lanza al mercado su gama de servidores de rango medio que se orientan al soporte de aplicaciones, Web, redes de área local, sistema de archivos de red, procesos de cálculo y bases de datos. Además soportan los procesos característicos de los entornos técnicos para EDA, MDA, energía e investigación científica. Al igual que la serie V, los modelos pertenecientes a la serie K, K370 y K570, se han diseñado para cubrir las necesidades en la realización de tareas de multiproceso simétrico y pueden incorporar desde uno a seis procesadores PA-8200, de 64 bits, a 200 mhz, con ancho de banda entrada/salida ampliable y memoria doble.

Estas características suponen un incremento en el rendimiento elevadísimo, ya que combinan su arquitectura con las capacidades de la tecnología de proceso paralelo masivo y multiproceso simétricos. HP también se centra en conseguir la escalabilidad necesaria para poder soportar procesos masivos en el entorno de la empresa. Así, mediante la arquitectura EPS se puede soportar hasta 16 nodos de multiproceso simétrico paralelo en un único sistema, permitiendo de esta forma la vinculación de procesadores para realizar la ejecución de una aplicación única en todo el sistema de la empresa. El rango de procesadores soportado está entre los 84 y los 512, según la gama.

HP en la empresa

Si comparamos la serie HP 9000 V2200 con otros productos en el mercado, podemos observar sorprendentes relaciones precio-calidad, con incrementos sobre el precio de entre un 53% y un 190% según los fabricantes. Así deben pensar importantes empresas del sector que apoyan estas tecnologías. Oracle, Andersen Consulting, Cap Gemini, Telefónica Sistemas o EDS son algunos ejemplos de empresas que utilizan HP en distintas implantaciones de sus sistemas.

Con esta serie de servidores, HP se adhiere a la tendencia de la informática distribuida, frente al mercado de los mainframes. Los principales clientes que HP tiene son organizaciones con mainframes costosos y lentos y empresas con necesidades concretas en aplicaciones para OLTP y DSS.

Linux X-WINDOW Window Manager, Visual JavaScript, TCL 7.6 y TK 4.2

CONTENIDO DEL CD

En este número hemos variado el entorno para facilitar el acceso a la información contenida en el CD; la aplicación utilizará el navegador que se encuentre instalado para visualizar los datos contenidos en el disco, la tendencia actual es que casi toda la información se facilite en formato HTML. Esto hace que parezca la manera más fiable de integrar datos, imágenes y sonido para que sean accesibles a todo el mundo con independencia de la plataforma donde estén ejecutando su navegador.

De este modo, ahora podremos visualizar el índice de las revistas actualizado para buscar algún contenido que nos interese sin necesidad de salir de la aplicación.

Esto también tiene sus inconvenientes, ya que al ejecutar la aplicación al mismo tiempo que el navegador del sistema, necesitaremos más memoria y en equipos pequeños no irá todo lo bien que debería.

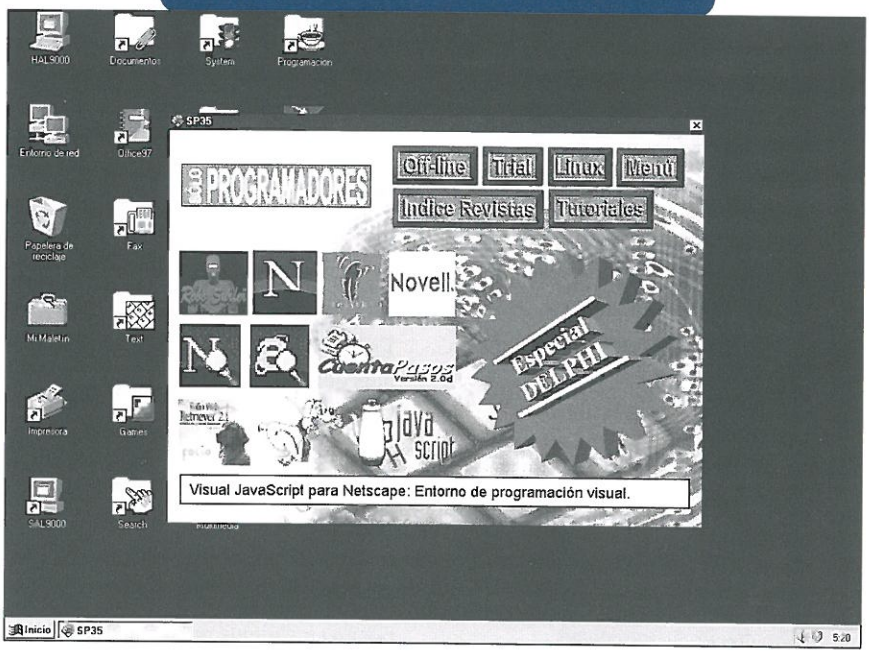
Navegación Off-Line

Aprovechando la facilidad de navegación hemos decidido incorporar al CD-ROM todos los meses alguna sede Web para la navegación off-line. ¿Qué significa esto? Muy sencillo: serán incluidos en el CD-ROM los lugares de Internet que consideremos de más interés para los programadores, de manera que sin necesidad de estar conectados (off-line) seamos capaces de viajar por ellos.

Esto le quita bastante emoción al asunto, aparte de que las sedes Web cambian continuamente y son actualizadas casi a diario. Sin embargo, servirá a la gente que no está conectada y nos ahorrará una cantidad muy grande de tiempo y dinero de navegación. Debido a que esta incorporación es de última hora, sólo hemos incluido la página Web de Optimax, pero prometemos que la oferta se ampliará en próximos números.

Ahondando en el mismo tema, también hay que decir que la inclusión de estas sedes no es tan fácil como parece. En primer lugar, el volumen de información que bajar es prohibitivo, además de que actualmente la red está plagada de nuevos protocolos, como Java, JavaScript y los consabidos módulos Active-X. Así, el download de algunos de estos sites, como podría ser el que tiene Microsoft, es

FIGURA 1



un verdadero trabajo de análisis y su inclusión en CD-ROM casi imposible.

Surf Agents

Otro elemento innovador es la utilización de ROBOTs en la búsqueda y recuperación de la información en Internet. Éstos facilitan la tarea de la revisión página por página de los datos sacados y depuran el contenido. Algunos, como el Web Retriever, son capaces de generar informes en formato de texto de la información. La mayoría de estas aplicaciones son programables de forma que permiten la utilización de los archi-conocidos motores de búsqueda de la red, como podrían ser los famosos Yahoo o Web-Crawler.

Este tipo de herramienta puede servir para realizar la tarea de la que hablábamos antes, que es la de poder cargar de una vez toda la información contenida en un Web.

Nosotros hemos incluido tres ejemplos de estas herramientas:

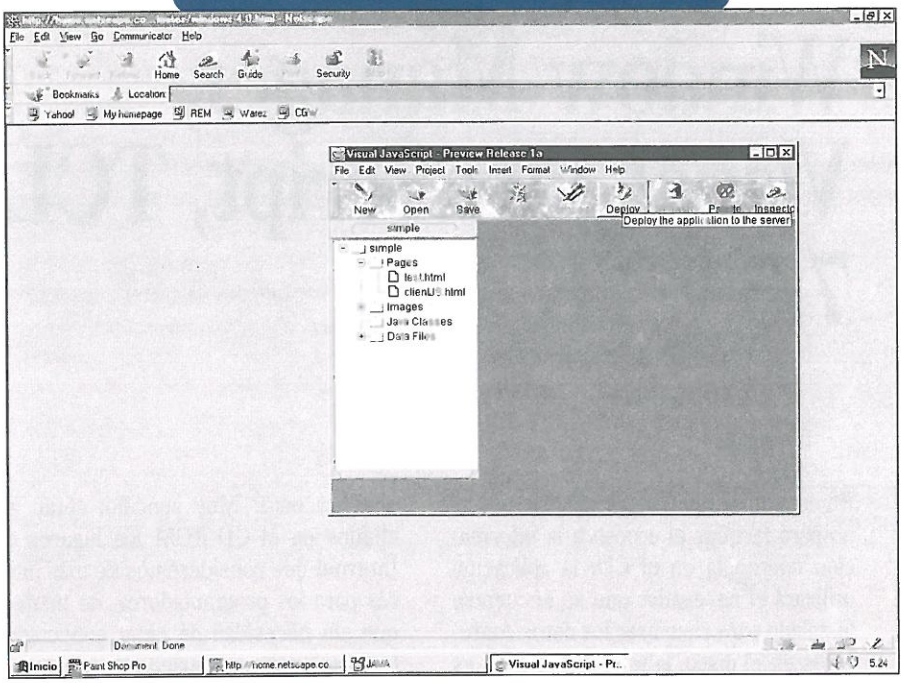
- Folio Web Retriever 2.1
- Go-Get-It v2.0
- Robo Surfer

Navegadores de Caché

Herramientas verdaderamente útiles, sirven para inspeccionar la caché cuando ya no estamos conectados. De esta manera podemos realizar un salvado de los datos, gráficos y documentos de los sites que visitamos sin perdernos en la amalgama de ficheros que forman la caché.

Hemos incluido un navegador de caché en su versión de Netscape y otro realizado para Internet Explorer, que son los más utilizados en la actualidad.

FIGURA 2



Visual JavaScript

Netscape nos facilita el aprendizaje y utilización de este lenguaje de programación con el entorno que nosotros facilitamos en este CD-ROM.

Linux X-Window Managers

Otra demostración de la versatilidad del Unix, y en concreto del Linux, es la proliferación de entornos gráficos para X-Window, algunos verdaderamente impresionantes y de los cuales incluimos en este número tres, **fvwm95**, **Enlightenment** y **Kde**, pero prometemos que habrá más.

Tutoriales

Sabemos que os gusta aprender y por eso incluimos mensualmente los tutoriales más populares en la red, para que os aclaren las dudas que podáis tener en diferentes temas, además en formato HTML. Este mes podéis disponer de un tutorial de JavaScript, para completar la información de la sección que aparece en esta revista.

Fuentes e Índice

Como viene siendo habitual, incluimos los códigos fuente que se mencionan en los artículos y el índice en formato HTML en el que es posible encontrar los contenidos de la revista.

Herramientas Delphi

Completando el especial sobre las herramientas de programación, podéis encontrar programas muy utilizados, que comentamos en profundidad en el primer capítulo dedicado a Delphi.

Fíjese. Si quiere echar un vistazo a los anuncios de DB2 y software para NT, visite nuestra galería publicitaria:
www.software.ibm.com/sw-sell/besteam/ads

DB2, Business Partner y BESTeam son marcas de IBM Corp. Windows NT es marca de Microsoft Corp. © 1997 IBM.

Forme parte de la red de Business Partners de IBM y podrá participar en proyectos de marketing como el BESTeam. Un programa internacional que apoya la comercialización de software de IBM. Además, se les proporciona soporte comercial y técnico, acceso a todo tipo de información sobre tecnologías de software, productos en demostración, condiciones especiales para formación y certificación en software de IBM. Si quiere aprovechar las ventajas de formar parte de la red de Business Partners de IBM o de ser miembro del programa BESTeam, llame al 900 100 400. Visítenos en Internet en:

www.europe.ibm.com/getdoc/psmemea/progserv/besteam/

www.ibm.es

IBM

Soluciones para nuestro pequeño mundo